NASA CR-163117

NASA-CR-163117
19830004843

ADVANCED FLIGHT CONTROL SYSTEM STUDY

G. L. Hartmann, J. E. Hall, Jr., E. R. Rang,
H. P. Lee, R. W. Schulte, and W. K. Ng

Contract NAS4-2876
November 1982

NASA

NASA CR-163117

ADVANCED FLIGHT CONTROL SYSTEM STUDY

G. L. Hartmann, J. E. Wall, Jr., and E. R. Rang
Honeywell Systems and Research Center
Minneapolis, Minnesota

and

H. P. Lee, R. W. Schulte, and W. K. Ng
Lockheed-California Company
Burbank, California

Prepared for
Ames Research Center
Dryden Flight Research Facility
under Contract NAS4-2876

National Aeronautics and
Space Administration

1982

Use of trade names or names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

CONTENTS

CONTENTS (continued)

CONTENTS (concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

ADVANCED FLIGHT CONTROL SYSTEM STUDY

G.L. Hartmann; J.E. Wall, Jr.; and E.R. Rang*
H.P. Lee; R.W. Schulte; and W.K. Ng†

SUMMARY

This study defines a new fly-by-wire flight control system architecture designed for high reliability. Spare sensor and computer elements are included to permit safe dispatch with failed elements, thereby reducing unscheduled maintenance. This program also formulated a methodology capable of demonstrating that the architecture does achieve the predicted performance characteristics. This methodology consists of a hierarchy of activities ranging from analytical calculations of system reliability and formal methods of software verification to iron bird testing followed by flight evaluation. This study concludes with a section on interfacing this architecture to the Lockheed S-3A aircraft for flight test. This testbed vehicle can be expanded to support flight experiments in advanced aerodynamics, electromechanical actuators, secondary power systems, flight management, new displays, and air traffic control concepts.

SECTION 1--INTRODUCTION

1.1 OBJECTIVES AND REQUIREMENTS

In broad terms, the objective of this program is twofold. One objective is to define a new fly-by-wire (FBW) flight control system architecture that possesses the integrity required by future commercial applications. Future energy-efficient aircraft will require:

---

*Honeywell Systems and Research Center, Minneapolis, Minnesota.
†Lockheed California Company, Burbank, California.

(1) Reliable digital FBW control

(2) Electromechanical actuators

(3) All-electric secondary power technology

The redundant, self-checking architecture defined in this study achieves the first element and is compatible with developments in the second and third areas. A second objective of this program is to formulate a methodology capable of demonstrating that the architecture does achieve the required level of performance. This hierarchical methodology ranges from analytical calculations of theoretical system reliability and formal methods for verifying software to laboratory and iron bird tests and actual flight experiments. A commitment to the proposed level of structure and rigor will lead to a validatable flight control system.

The definition of an advanced digital fly-by-wire (ADFBW) architecture is a technology integration task. State-of-the-art assessments and trends in the underlying computer, sensing, and actuation areas were used to select from a number of design alternatives.

In later sections of this report, the Lockheed S-3A aircraft is discussed as a potential testbed vehicle. However, the ADFBW architecture was not developed specifically for the S-3A. A generic FBW system is assumed for an aircraft requiring three axes augmentation, gain scheduling based on air data measurements, and angle-of-attack limiting consistent with reduced static stability airframe designs. Therefore, a sensor suite will include pilot transducers, body rates and accelerations, and air data measurements.

The pacing requirements for all FBW systems are the reliability-related qualities of flight safety, mission reliability, and availability. Numerous programs have developed redundancy structures for both military and commercial applications that satisfy flight safety and mission reliability through various combinations of triplex and quad redundancy, all of which produce at least dual-fail-operative performance. The failure rates of current components indicate the necessity for considerable unscheduled maintenance. For example, the mean-time-between-maintenance for flight control sensors plus electronics will be approximately 250 to 1000 operating hours, depending on system complexity and design maturity. In a commercial application with 2000 hours every six months, a potential dispatch problem is evident.

The reliability of the advanced flight control system should be such that its loss would not be expected during the lifetime of a large commercial fleet of aircraft using it. This requirement, when reduced to a probability of loss per flight hour, produces a figure on the order of 10^{-9}. For example, assuming 10 flight hours per day produces 3600 hours per year per aircraft. A fleet of 200 aircraft operating for 15 years accumulates about 10^{7} flight hours. Allowing a 1% loss probability results in the 10^{-9} per hour figure.

The desired maintenance quality implies fault tolerance in excess of that needed for flight safety. Various strategies are conceivable for achieving this. Since flight safety is of first priority, no aircraft will be dispatched if the flight control status is not adequate. Consequently, the maintenance requirements may be posed in terms of an allowed probability of unscheduled maintenance over a given period. For this study the period was defined as six months, or about 2000 hours for a commercial aircraft. If this probability is low, then the system maintenance quality is adequate. It seems reasonable that a large percentage of a commercial fleet should not require unscheduled maintenance over the stated period, perhaps 90%. If such a level were achieved (a six-month unscheduled maintenance probability of 0.1), a dramatic improvement in maintenance quality of current FBW systems would be realized.

1.2 REPORT ORGANIZATION

This report is organized in eight sections plus two appendixes. Section 1 is the introduction. Section 2 presents the recommended architecture and implementation. Section 3 starts the discussion of the validation methodology by addressing system and interface specifications. Section 4 presents a method of reliability prediction. Section 5 concentrates on the software development process. Section 6 concludes the validation methodology by addressing the system test phase. Section 7 presents the S-3A interface to the recommended architecture and discusses the interface to the electrical and hydraulic systems. Section 8 presents conclusions and recommends development and flight test of the ADFBW architecture.

1.3 ACKNOWLEDGEMENTS

This study was the product of the efforts and diverse talents of a number of people working at Honeywell, Lockheed-California, and NASA. The authors wish to express their thanks to these people, especially to Mr. J.C. Larson of Honeywell's Avionics Division and Mr. R.L. Heimbold of Lockheed-California. Appreciation is also expressed to the technical management at the Hugh L. Dryden Flight Research Center, in particular to Mr. K.J. Szalai, Mr. A.F. Myers, and Mr. L.W. Abbott.

SECTION 2--DEVELOPMENT OF AN ADVANCED ARCHITECTURE

The ADFBW flight control architecture must exhibit ultra-reliability with low maintenance and must be validated and verified to a high degree of confidence. To achieve the ultra-reliability objective, redundant elements are used. The system must be able to tolerate multiple faults while maintaining undegraded operation. In designing this fault-tolerant system, reliability analysis plays a major role in the system architecture selection process. Section 2.1 summarizes design tradeoffs in terms of preliminary reliability and maintenance characteristics. A more detailed analysis of the recommended architecture is made using fault tree analysis in section 4. The generic ADFBW architecture is defined in section 2.2, based on our design objectives of ultra-reliability plus ease and confidence of validation. An implementation of the recommended architecture for the S-3A testbed is contained in section 2.3.

2.1 RELIABILITY AND MAINTAINABILITY ISSUES

Advanced flight control architectures are built on advances in the underlying sensor, computer, and actuator technologies. Honeywell participated in a study of 1990 flight control technologies as part of a study of integrated application of active controls technology to an advanced

subsonic transport (ref. 1). In this study, we assessed the trends in the following technologies:

(1) Sensors
- Air data
- Angular rate
- Accelerometers

(2) Airborne computer technology
- Instruction set and higher-order language trends
- Integrated circuit advances
- Buses (including fiber optics)

(3) Design and validation
- Flight control functions
- Formal specifications
- Software design and code
- Verification and validation

(4) Actuators
- Hydraulic power sources
- Electric power sources

An in-depth treatment of this technology status and its trends may be found in reference 1.

Several conclusions from this investigation are pertinent to the design of an ADFBW architecture. Hardware improvements will not remove the need for sensor redundancy. Reliable sensing can be achieved through sensor redundancy and an increased use of the computer system. Present aircraft actuation systems use redundant hydraulic elements to achieve sufficient reliability for FBW requirements. New actuator developments are aimed at improving efficiency through the use of electromechanical actuators. The ADFBW architecture must be compatible with both types of actuation systems. Significant advances in computer hardware are expected through developments in large-scale integrated circuit technology. Software costs are expected to continue to dominate hardware costs in DoD/NASA applications. This trend emphasizes the need to carefully trade off whether a particular system function is to be performed in hardware, software, or some combination of both.

The reliability of typical components for a flight control system is shown in table 1. These values are projected for the mid-1980's. A single string of sensors, computer, and hydraulic actuator--as illustrated in figure 1--has a mean time between failures (MTBF) of approximately 1500 hours, orders of magnitude less than our requirement. Hence, replication of sensing, computing, and actuation elements is mandatory.

Next consider a triple set of these elements and assume that the redundancy management allows operation with only one of the three channels functional. If there is no crossfeeding of sensor and computer to the actuation, the unreliability is about 0.28×10^{-9} at one hour corresponding to failure of the three channels. If the redundant elements are fully crossfed, the number of success paths increases. In this case the unreliability decreases by more than a factor of 20 to 0.013×10^{-9} at one hour. These trends are shown in figure 2, which illustrates that redundant elements with crossfeeds improve overall system reliability.

TABLE 1. - RELIABILITY CHARACTERISTICS OF FLIGHT CONTROL ELEMENTS

Element	Failure Rate ($\times 10^{-6}$)	Comments
Air Data Computer	91.	Three-year extrapolation of existing products
Pilot Transducers	40.	Pitch, roll, yaw sensors* plus A/D electronics
Rate Gyros	30.	State of the art
Accelerometers	30.	Precision floated pendulum or quartz fiber
Serial Data Buses	10.	Estimate based on chip count
Computer	200.	Estimate of self-checking microprocessor
Actuator	90.-140.	Based on state-of-the-art (includes electronic and hydraulic components)

*These position sensors could be a linear variable differential transformer (LVDT).

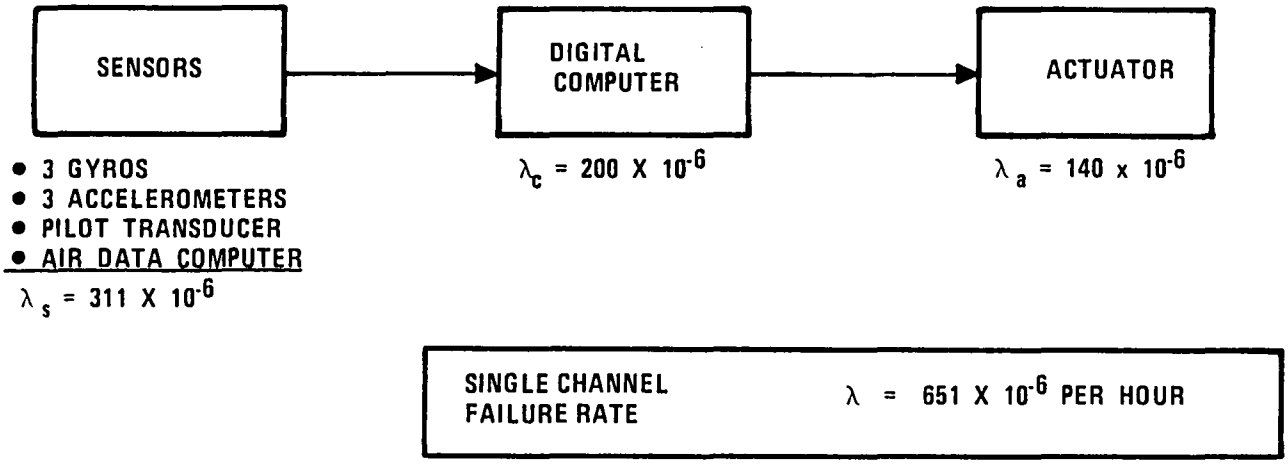


Figure 1. - Flight control elements--single string.

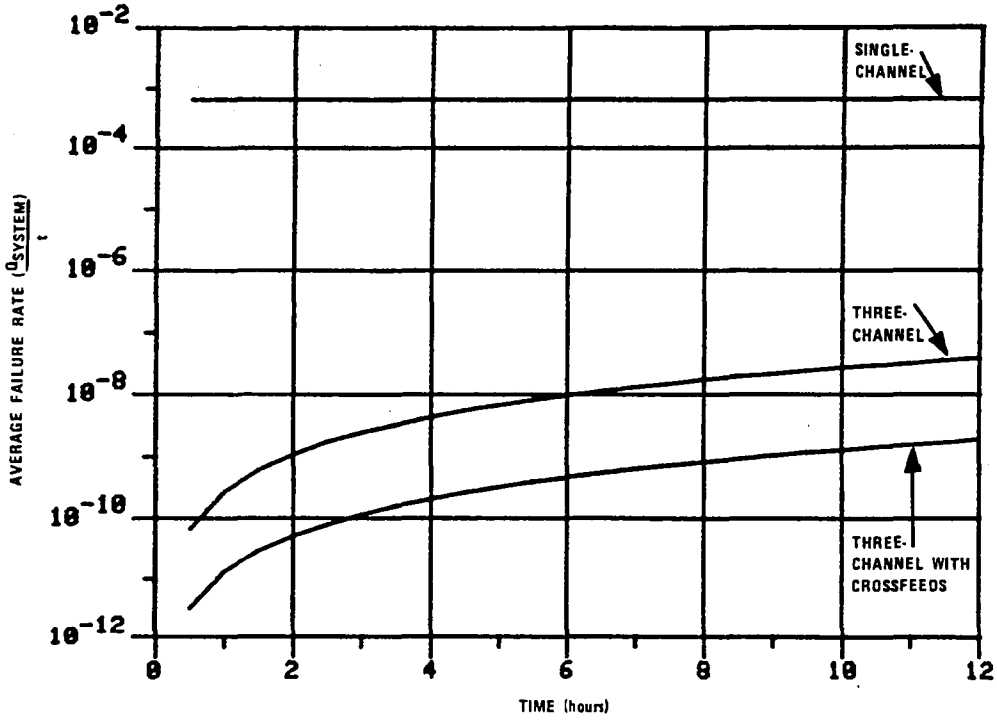


Figure 2. - Reliability trends of redundant flight control systems.

Sufficient redundancy must be provided to meet both the flight safety requirement and the low maintenance goal. As explained in section 1, the goal of low maintenance is taken to mean that the probability of unscheduled maintenance is less than 10% over 2000 flight hours. Thus, two distinct reliability issues enter the design process:

- Probability of catastrophic failure ($<10^{-9}$/hr)
- Probability of unscheduled maintenance (<0.1/2000 hrs)

Both probabilities impact the recommended architecture.

For design purposes, it is useful to apportion these two probabilities among the various components of the flight control system. This is somewhat analogous to the well-known error budgeting process. The probability of catastrophic failure is apportioned roughly evenly between the sensor suite, computers, and actuators. For the sensors, this probability is further broken down into contributions from air data computers, pilot transducers, and inertial sensors. The probability of unscheduled maintenance is about evenly divided between sensors and computers. This is because it was impractical to reduce the probability of unscheduled maintenance for the actuators to near 10% per 2000 hours.

The next sections will summarize the design decisions in the organization of the redundant sensing, computing, and actuation elements. In selecting the recommended architecture from the various design alternatives, experience with other systems and interpretation of the technology trends play a large role in weighting the benefits of one approach against another.

2.2 RECOMMENDED ARCHITECTURE

2.2.1 Overview

The recommended flight control system architecture is shown in figure 3. This advanced, self-checking architecture is capable of meeting the flight-critical safety requirements and the goal of low system maintenance. Further, the proposed structure facilitates verification and validation of the system's performance.

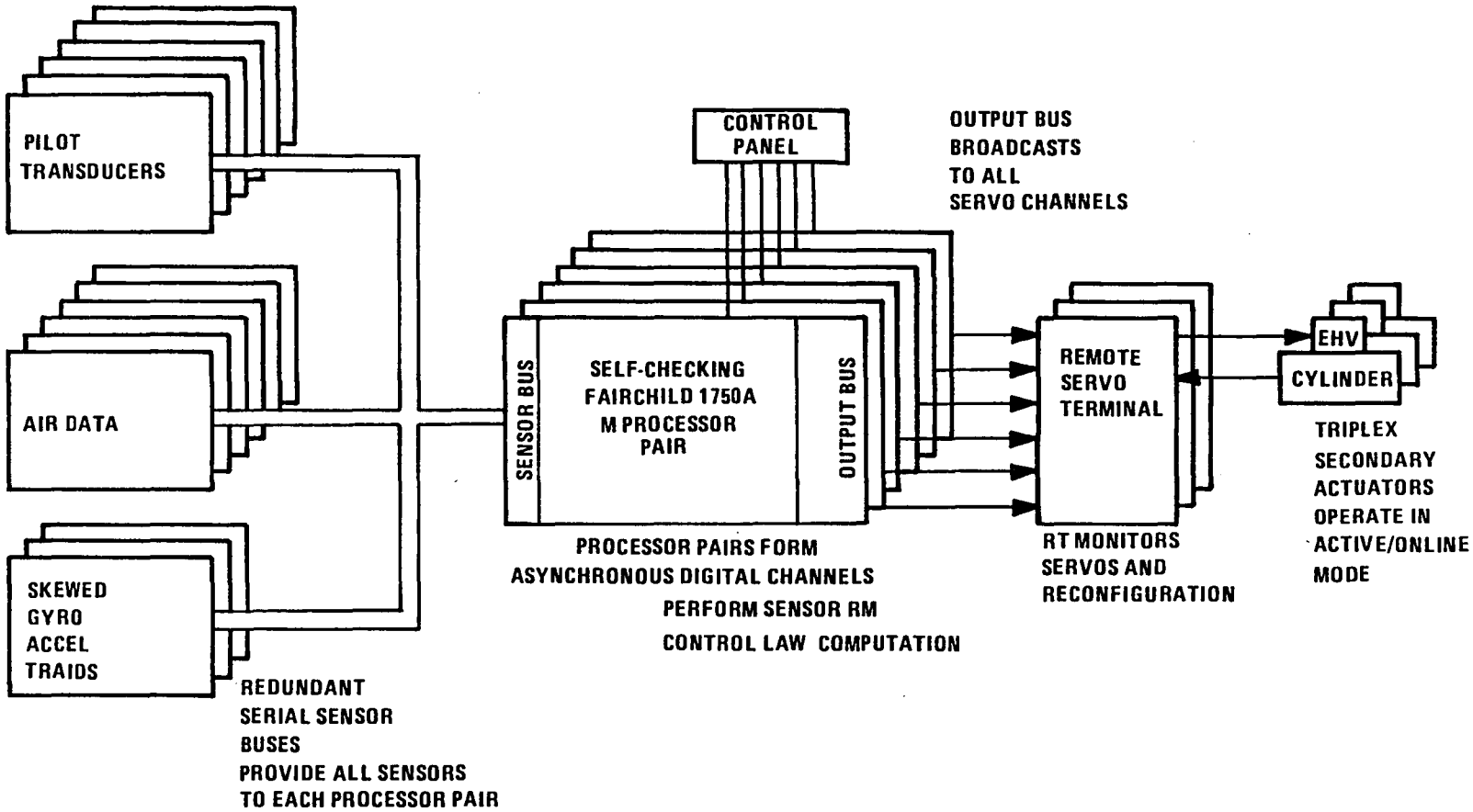


Figure 3. - Overview of ADFBW architecture.

The baseline suite of sensors consists of six sets of pilot input transducers, six air data computers, and three skewed triads of inertial sensors. Spare sensors are treated as cold spares and reconfigured only during preflight. The sensors interface with the computer channels via six serial sensor buses. Six parallel computer channels of self-checking microprocessor pairs are proposed. Spare processors operate as hot spares. The bus controller for each sensor bus is included within a computer channel. The computers broadcast over redundant command buses to triplex servo terminals. The number of such triplex servo terminals is dependent on the specific application. The remote terminals are compatible with either conventional hydraulically powered actuators or electromechanical actuators.

The following subsections provide more detail on the individual elements of this architecture:

Sensors

Computers

Actuators

Figure 4 provides a summary of the total unreliability for the ADFBW architecture. This curve is obtained by summing the sensor, computer, and servo contributions and excludes electric and hydraulic power sources. The individual reliability characteristics are developed below. A more detailed analysis of reliability using fault tree modeling is presented in section 4.

2.2.2 Sensors

The basic control mode in this study requires pilot input transducers, body rates and accelerations, and air data. The initial assignment of sensor redundancy was made based on flight safety considerations. For pilot input and air data sensing, replication to the desired level of redundancy to permit dispatch with failed elements is required. For the rates and acceleration measurements, a skewed sensor assembly is recommended.

Air data. - The air data computers provide angle of attack plus the usual air data derived quantities. Current production air data systems are expected to achieve, within three years, a MTBF of 11 000 hours. Hence, the failure rate is 91×10^{-6} per hour (table 1).

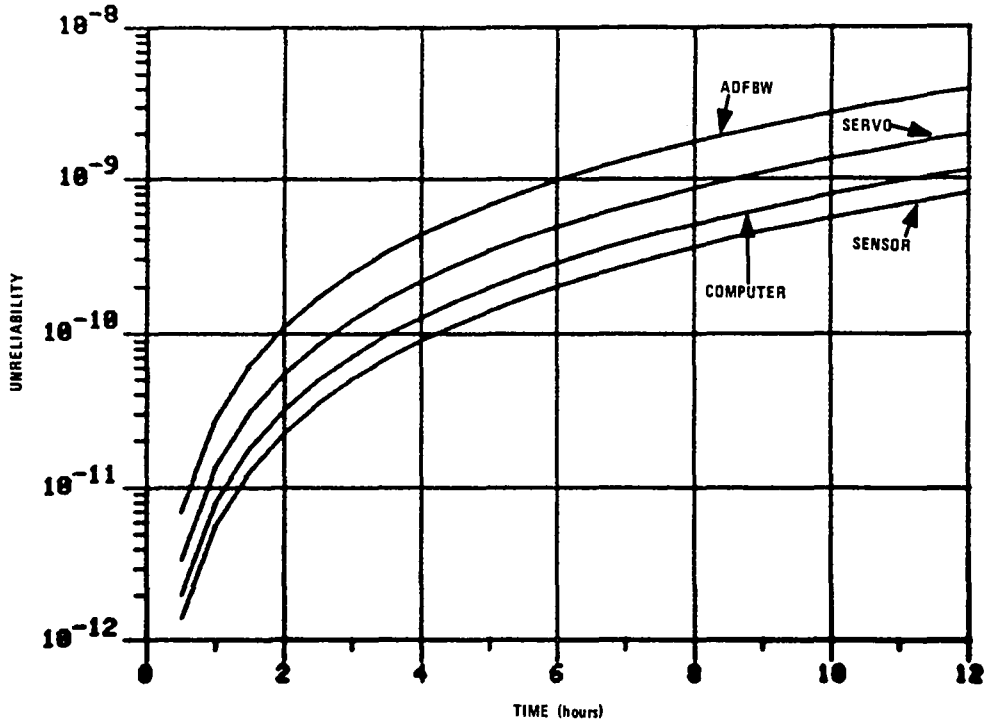


Figure 4. - ADFBW reliability predictions.

Estimates of the probability of loss of air data are used to decide the minimum level of redundancy for safe dispatch. Comparison monitoring is used to detect and isolate failures of the air data computers. With quad redundancy, a loss of air data occurs when three of the four units have failed. This probability is estimated as

$$Q_{\text{air data}} = \binom{4}{3} Q^3 \ (1 - Q)$$

$$= 0.048 \times 10^{-9} \text{ (averaged over 4 hours)}$$

This unreliability is plotted as a function of flight duration in figure 5. A triplex system loses air data when two of three units fail, and this probability is too high. Therefore, the following requirement for safe dispatch is established:

At least four air data computers must be operating for dispatch.

Spare air data computers are supplied in order to satisfy the maintenance objective. Maintenance of the air data computers is required when the number of failed units exceeds the number of spares. The recommended architecture has six air data computers--that is, two spaces are provided. Maintenance is required when three of the six units fail preflight checks. The probability of this event is shown in figure 6. At 2000 flight hours, the probability of unscheduled maintenance on the air data computers is about 5%.

Pilot input transducers. - The pilot input transducers include pitch and roll stick and pedal transducers. The transducers are LVDTs. The pilot input terminal includes the electronics for accepting three axes of commands, performing A/D conversion, and interfacing with the sensor bus. Table 1 shows the expected failure rate to be 40×10^{-6} per hour.

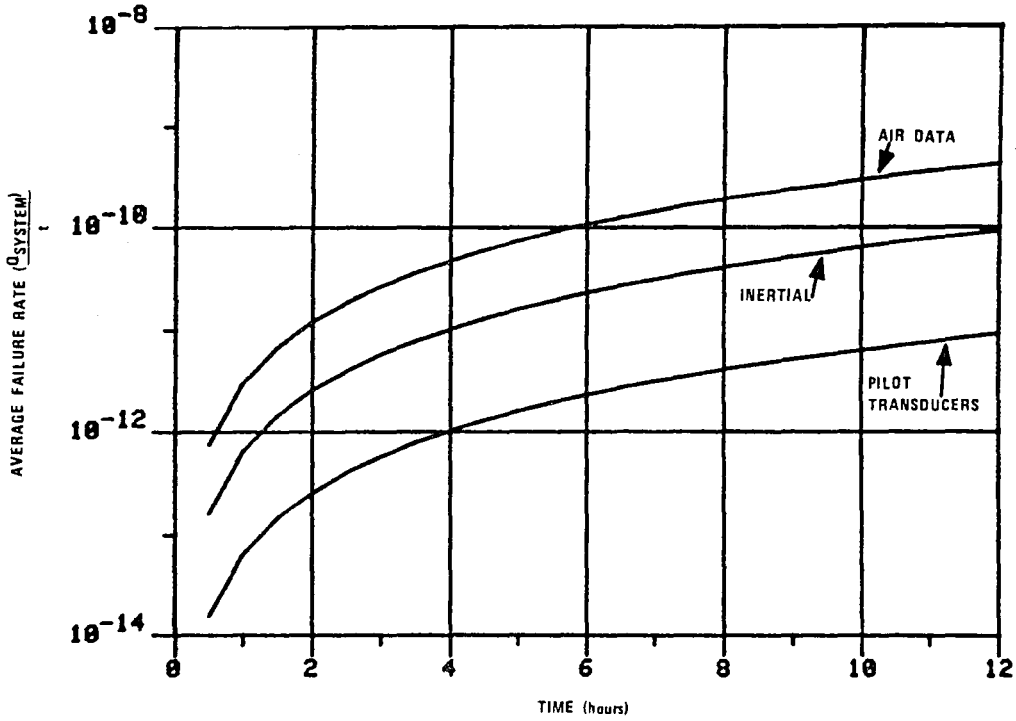


Figure 5. - Sensor unreliability trends.

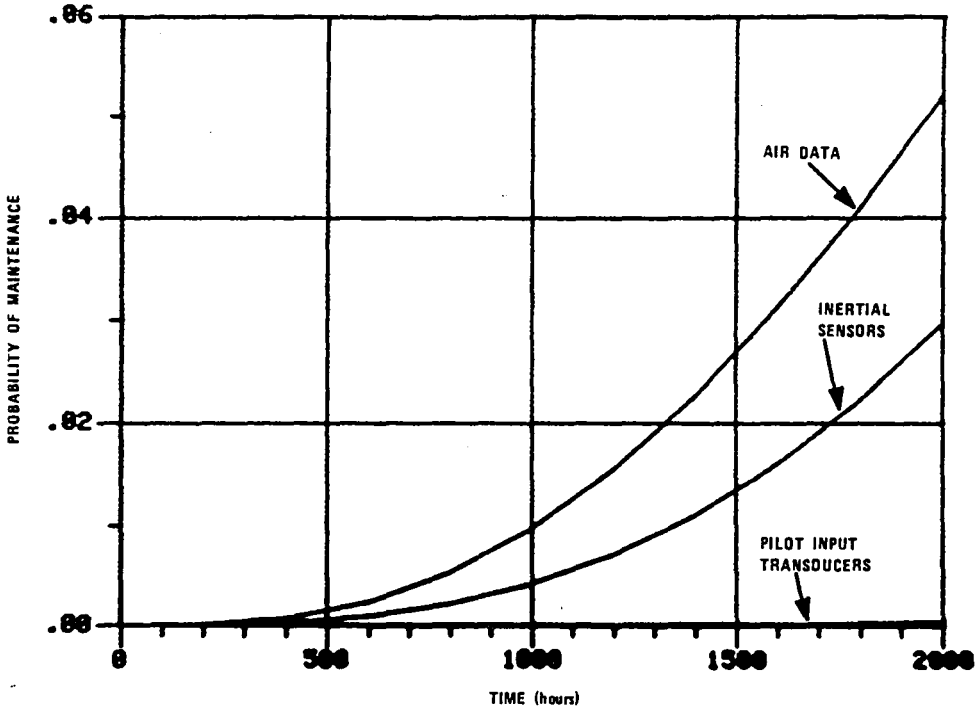


Figure 6. - Sensor maintenance trends.

The self-test coverage of these devices is 100%. This means that operation with only one unit is possible. With a triplex system, the probability of loss of pilot sensors equals the probability of three failures out of three units.

$$Q_{PI} = Q^3$$
$$= 1.02 \times 10^{-12} \text{ per hour (averaged over 4 hours)}$$

This probability is plotted in figure 5, and is negligible. The probability of loss of pilot sensors with only a dual system, however, is too great. This leads to the requirement that

At least three pilot sensor terminals must be operating for dispatch.

Once again, maintenance is required when the number of failed units exceeds the number of spares. With the six units included in the architecture, three spares are provided, and maintenance is required only after four failures. The probability of unscheduled maintenance on the pilot input transducers is plotted in figure 6 and is shown to be less than 1% after 2000 hours.

Inertial sensors. - Body rate and acceleration measurements are required in three axes. The failure rate for each of these six sensors is 30×10^{-6} per hour (table 1).

Comparison monitoring is used to detect and isolate failures of the inertial sensors. This scheme results in the loss of inertial sensing when all but one of the sensors of any type fail. For a quad-redundant system, the probability that three of four sensors fail is simply

$$Q_{IS} = \binom{4}{3} Q^3(1-Q)$$
$$= 0.01 \times 10^{-9} \text{ per hour (averaged over 4 hours)}$$

Figure 5 shows this reliability as a function of flight duration. A triplex system has an unacceptably large probability of loss of inertial sensing, so for safe dispatch we require that

At least four inertial sensors of each of six types must be operating.

The three components of figure 5 were combined to yield the sensor curve shown previously in figure 4.

By providing two spares, maintenance is required when three sensors of any type have failed. The probability of this event is plotted in figure 6. A 2% probability of unscheduled maintenance for the inertial sensors occurs at 2000 hours. This suite of gyros satisfies our reliability and sparing goals. However, it does involve a large number of sensors:

Rate gyros: 3 axes x 6 = 18 total

Accelerometers: 3 axes x 6 = 18 total

In order to achieve some reduction in hardware, we recommend skewed sensors. Two skewed triads (six sensors) can provide dual-fail-operational capability. Three skewed triads provide sparing and can permit safe dispatch with up to three sensors failed. Basically, sparing is accomplished by configuring a hexad arrangement from the nine sensors as part of a preflight check. The two triads are oriented such that each axis of the triad makes an equal angle with the aircraft x-axis and no three axes of any set are co-planar.

Skewed sensors are presently receiving increased attention (ref. 2, 3, and 4). Skewed sensors require higher resolution and dynamic range than conventional configurations and must avoid saturation. It is felt that gyro

and accelerometer technology has advanced such that the additional demands placed on sensors if they are skewed can be satisfied. Honeywell is currently working three major areas involving skewed sensor assemblies:

(1) Multifunction control reference system. - This is an Air Force program to flight test on an F-15 hexad (two orthogonal triads) composed of six accelerometers and six ring laser gyros. The "boxes" consisting of sensors plus computer are skewed (ref. 2 and 3).

(2) Integrated sensor assembly. - This Navy program uses a different architecture. Five boxes are used--two for integrated sensor assemblies, three for flight computers. One computer is dedicated to the strapdown navigation computations; the other two perform flight control. In this architecture only the sensor assemblies are skewed. Flight test is planned on an F-14 or F-18.

(3) Integrated inertial reference assembly. - This Air Force program is in a study phase with the objective of defining future skewed sensor requirements for the 1990's.

With the commitment to skewed sensors in the aircraft community, we feel such a sensor approach is viable for our architecture and provides an elegant solution to the inertial sensor redundancy problem.

Sensor redundancy management. - The redundancy management of the sensors is replicated in each digital channel. Each processor can independently detect and isolate sensor faults. During preflight, a sensor suite will be identified. Redundant sensors in excess of the dispatch requirements will be ignored. This eliminates the need for mid-value selecting from up to six signals. The redundancy levels required are:

(1) Pilot transducer terminals
   - Three terminals

(2) Air data computers
   - Four compares

(3) Skewed gyros/accelerometers
   - Two triads

The pilot input transducers and air data quantities use conventional comparison monitoring. Since the digital channels operate asynchronously, data skew between the signals being monitored is to be expected. This data

skew will result in a time delay (less than one-half the sampling period) for mid-value selected signals. Sampling rates will be selected so this delay does not compromise phase and gain margins.

Management of the skewed sensors would be based on the use of parity equations (ref. 2). Parity equations take advantage of the fact that sensor skewing provides redundant information. Basically, the outputs of any four sensors may be linearly combined (via direction cosines that describe geometry) to form a parity equation. In the absence of sensor errors the parity equations equal zero. The number of parity equations available for sensor redundancy management is computed from N sensors taken four at a time. Thus, six sensors yield 15 parity equations.

The multifunction control reference system skewed assembly is being applied to a high-performance fighter and the sensor triads have been separated to demonstrate the concept in a "worst-case" installation. Thus compensation for various moment-arm effects has been included to allow low decision thresholds. In addition, effort has been expended to tailor parity equations for real-time use (ref. 2). At each sample time, subsets of the parity equations are used based on a table look-up calculated from the failure status of the previous pass. This approach yields the following advantages:

(1) Minimum processor usage (redundancy management decisions are computed off-line and stored in look-up tables)

(2) Ability to deal with dual simultaneous failures

(3) Noise immunity (via the use of trip levels)

(4) Flexibility (look-up tables and trip levels are easily adjusted)

(5) Two-level operation--acceptable sensitivity without false alarms

(6) Decisions based on the status of all parity equations computed

Skewed sensor technology has made significant progress. Our architecture will be able to integrate this technology to provide cost-effective sparing of the inertial information.

2.2.3 Computers

The recommended computer system consists of six redundant, parallel computer channels. Each channel is a self-checking microprocessor pair that implements all of the flight control modes. Each self-checking pair listens

to all the sensor data on each of the six sensor buses. Each channel independently performs sensor selection and control law computations. Each pair also transmits, on the sensor bus it controls, various integrator values for cross-channel equalization. This transmission is necessary because the six computer channels run asynchronously. The software to perform these functions is estimated to require a computer loading of less than 400 kops.

The major design alternatives considered are:

(1) Distributed processing with spares versus redundant channels performing identical tasks

(2) Single or multiprocessor channels

(3) Fault detection and isolation

(4) The number of channels required to achieve high reliability and provide sparing for low maintenance

Each of these areas is discussed below.

The first design alternative involves options in defining a philosophy for managing the computers. Two options are:

(1) A distributed system with an operating system that distributes tasks among healthy processors. This approach requires the distributed system to reconfigure to remove failed processors. Examples of such systems are given in references 5 and 6.

(2) A fixed tasking structure with a planned fallback to redundant copies of tasks running in parallel in separate processors. This option uses redundant channels of computers. Many current systems are implemented in this manner.

The second option is a simpler alternative. Its software is far less complicated and its operation in response to failures is easier to verify. This alternative must be favored in an architecture seeking ultra-reliability and complete validation, particularly if the redundant channels are single processors.

The second design issue is whether a channel should be implemented as single or multiple processors. We believe that valid software can be produced for either arrangement by proper definition of software modules and control of their interfaces. Historically, problems arise when the processors start to become heavily loaded and modules are compromised to "fit everything in." We

have estimated the software load for a generic FBW system, and have concluded that with advanced microprocessor technology, a single processor will be less than 50% loaded. In addition, our architecture has been defined to eliminate and simplify software, especially in the redundancy management functions. Thus, we are led to recommend the simple alternative of performing all the computations in a single processor. As will be explained in the next section, the selected processor is readily expandable to a multiple processor configuration should the throughput requirements grow.

The next issue involves the fault-tolerant operation of the multichannels. The two major design alternatives are majority voting and self-test. Either option may be implemented in a hardware-intensive or software-intensive fashion. To date, most redundant flight control computers have used voting to detect first faults, since software self-test of processors has never claimed 100% coverage.

In past designs some computers have used duplication of key circuits as a way of providing self-test. With today's microprocesor technology it is attractive to duplicate single-chip processors to provide complete coverage. We have been developing such an approach for the Air Force since 1978 (ref. 7). The self-checking microprocessor pair (SCMP) is a processor with 100% self-test via hardware duplication at the integrated circuit (IC) level. Various tradeoffs have determined that if a SCMP fails, it should be removed from the system. No attempt is made to identify the healthy half and pair it with a spare processor to form a new SCMP. The latter option is possible, but was found to compromise the simplicity of the SCMP design. We recommend self-test as implemented in the SCMP.

The remaining design issue concerns the number of parallel channels needed to satisfy reliability and low maintenance demands. Figure 7 shows a plot of the failure probability of a triple-channel configuration of SCMPs. This curve appeared previously in figure 4. This arrangement is dual fail-operational and allows operation with one digital channel. The probability that the three digital channels fail is less than 0.13×10^{-9} per hour (averaged over a four-hour flight). For safe dispatch, we impose the following requirement:

That at least three SCMPs must be operating.

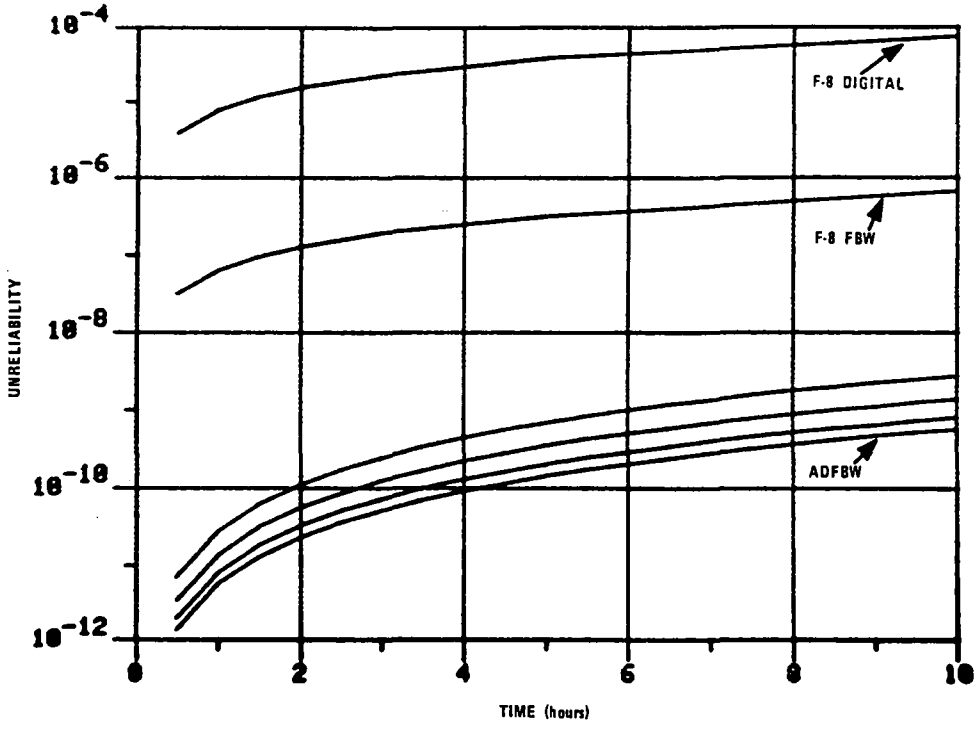


Figure 7. - Unreliability trends in digital channels.

Figure 7 also shows the unreliability of the F-8 digital system and the F-8 FBW system (digital plus analog backup). The ADFBW offers a significant improvement by virtue of MTBF advances due to very high-speed integrated circuit (VHSIC) technology and by permitting single-channel digital operation due to self-checking hardware.

Maintenance trends are shown in figure 8, which compares maintenance probabilities if first failures must be fixed. Curves for computers with a MTBF of 1450 hours (F-8 DFBW) and 5000 hours (SCMP) are shown. If three spare SCMPs are provided, then maintenance is required after the fourth failure rather than the first. In this recommended situation, the probability of unscheduled maintenance for the computers is less than 8% for 2000 hours of operating time.

2.2.4 Actuators

This subsection defines the elements of the architecture that interface the computers to the surfaces. Present aircraft have a hydraulic power

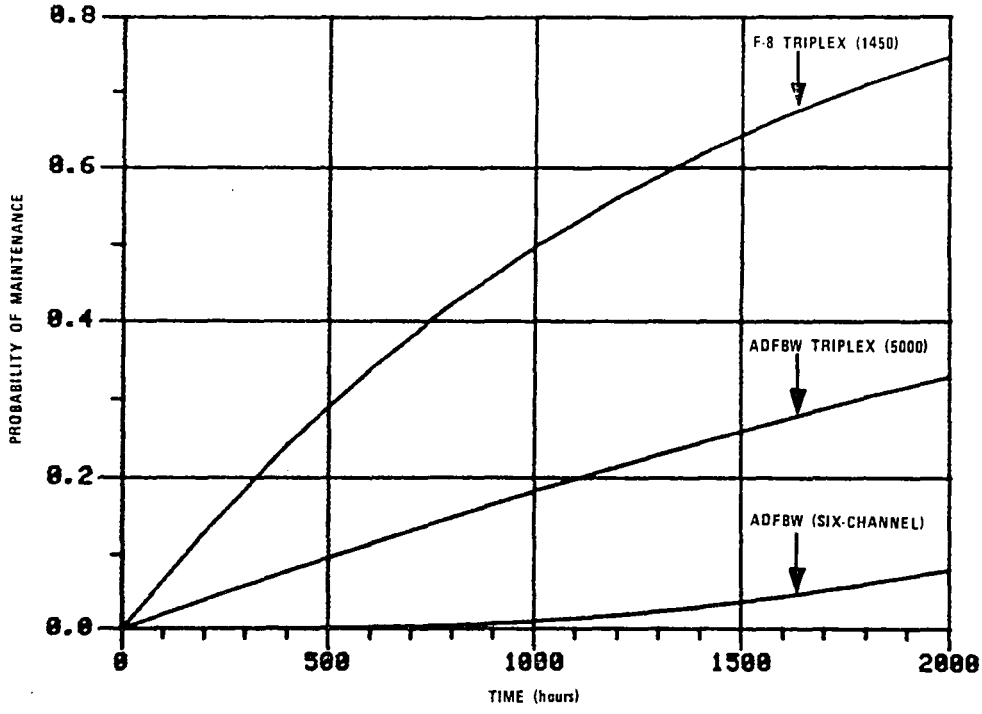


Figure 8. - Maintenance trends in digital channels.

generation and distribution system that has evolved over many design cycles. It is reliable and relatively easy to maintain. We recognize, however, that the long-term trend will be to use electromechanical actuators for energy efficiency. At this time, electromechanical actuators are not sufficiently mature to be considered for the ADFBW architecture. The proposed implementation uses conventional hydraulically powered actuators.

There are advantages in defining a simple, clean interface between the digital channels and the servo electronics. Our recommendation is to use a remote terminal that provides this interface via a digital serial bus. The remote terminal can provide all the redundancy management and reconfiguration required of the servos. This simplifies the computer software since it is not involved in the servo loop closing or redundancy management. The remote terminal can be simply specified, built, and tested. The remote terminal commands surface position and provides the loop closure for each servo actuator. Details of the remote terminal are given in section 2.3.

For this application triple-servo channels and actuators are used. This is consistent with our baseline testbed, the S-3A. Servo monitoring is provided to allow operation down to one channel. The projected unreliability for a triple-servo channel was plotted previously in figure 4 as one of the components of the ADFBW reliability. The assumed failure rate includes both electronic and hydraulic components (table 1). The failure probability is 0.22×10^{-9} per hour (averaged over a four-hour flight). Thus, it is not safe to dispatch with a failed servo channel and obtain a failure rate less than 10^{-9} per hour.

Maintenance actions are required after a first failure. This yields the probability of unscheduled maintenance shown in figure 9. A range of probabilities is shown for a triple channel with a servo-channel MTBF between 7140 hours and 11 000 hours. If a fourth servo channel were provided such that maintenance occurs after the second failure, there is still a 60% probability of unscheduled maintenance over 2000 operating hours. The servo channel MTBF would have to improve by another factor of three to 33 000 hours before the probability of unscheduled maintenance approaches 10%. Based on the cost and difficulty of replicating hydraulic actuation of the various surfaces, a triplex arrangement is recommended.

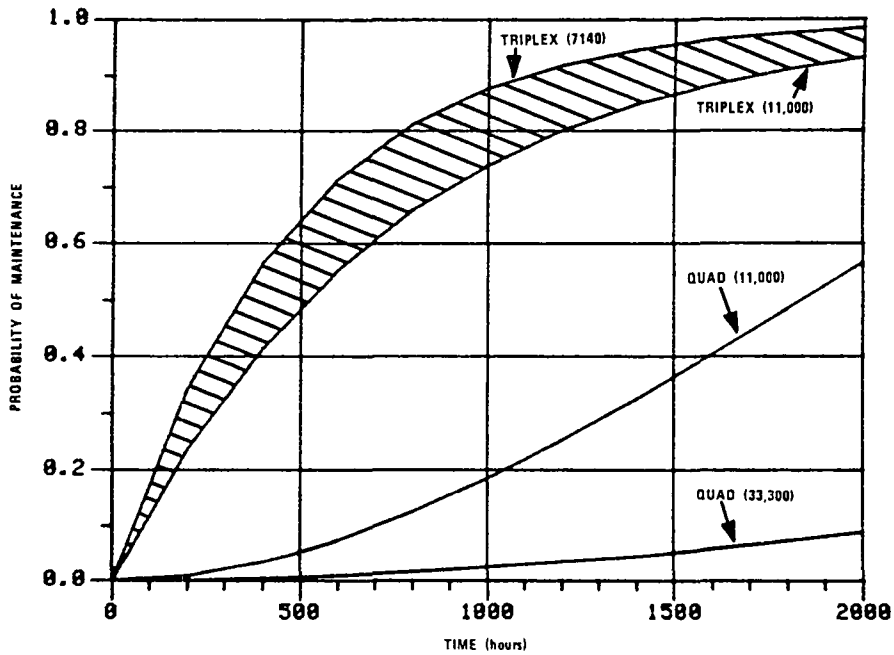


Figure 9. - Maintenance characteristics of servos.

2.3 RECOMMENDED IMPLEMENTATION

The key features of the recommended advanced, self-checking architecture are:

(1) Skewed gyro/accelerometer sensor assemblies

(2) Redundant serial buses to provide all sensor data to each computer

(3) Redundant, self-checking microprocessor pairs

(4) Serial command buses to remote terminals

(5) Remote terminals to provide command distribution and redundancy management of each hydraulic servo actuator

This section provides details on a recommended implementation of this architecture for the S-3A. Particular attention is given to the servo remote terminal as an illustration of self-checking approaches and the overall design methodology of section 3. This section emphasizes the key features and does not go into detail on preflight testing and other built-in tests that would be a part of the final detailed design.

2.3.1 Sensors

Six pilot input terminals and air data computers are included in the architecture. The proposed implementation uses standard LVDTs and air data computers. Each terminal and air data computer interfaces with one of the six sensor buses.

The body rate and acceleration mesurements are derived from three skewed triads. The individual sensor data in these triads will appear on redundant buses to reduce the effects of bus failures. For example, each sensor could be read out on four of the six buses.

The rate gyro is a Honeywell-developed ring laser gyro. The laser gyro output is a 9600 Hz digital signal consisting of a string of pulses, each representing an increment of integrated aircraft angular rate. These pulses are summed in an accumulator to provide angular displacement over the preceding 0.02-second sample interval. This quantity is converted to a signal proportional to angular rate.

The accelerometers will also be high-quality instruments, compatible with skewing requirements. Candidates include either floated pendulum devices (Sundstrand QA 2000, Donner 4852) or quartz fiber devices (like Honeywell's GG326).

Serial data bus. - Six independent serial buses were shown in figure 3. The bus controller logic is packaged with the digital computers. However, it will continue to manage the bus after a SCMP fails. A 1553B type of protocol has been assumed, whereby the sensors supply data in a command/response mode. There are some advantages in using 1553B since it is a standard, and several manufacturers are offering chip sets that provide the interface. These hardware elements permit a high degree of redundacy without an inordinate hardware (size, weight, power) penalty.

A recommendation to use wire or fiber-optic buses has not been made. Ultimately, fiber optics will be used. It may be appropriate to use wire in earlier phases.

2.3.2 Computers

This architecture has been built around SCMPs. Six channels of SCMPs run asynchronously. For protection from hazards, they should be packaged as two channels per 1/2 ATR box. We have identified the Fairchild 1750A as the leading candidate microprocessor. This selection is based on several considerations:

(1) In view of today's processors, the Fairchild 1750A offers a dramatic improvement in throughput. This offers the opportunity to do all the flight control software in a higher-order language (HOL) in a single processor and still have room to grow. Its availability is consistent with the schedule of this program.

(2) 1750A is an Air Force standard, and may evolve to the Joint Services standard for 16-bit machines. By virtue of its being a standard, there is a commitment to the development of extensive support tools, maintenance of a HOL, etc. In contrast, NASA developed its own HOL for Shuttle, and as the only user has had to provide all the maintenance and compiler upgrades.

The Fairchild 1750A is a 3-micron, isoplanar integrated injection logic (I^3L) device, based on the Fairchild 9445. Its characteristics are summarized in table 2. The logic symbol showing the pins and their functions is shown in figure 10, taken from reference 8.

TABLE 2. - FAIRCHILD 1750A CPU FEATURES

o Single 64-pin microprocessor implements MIL-STD-1750A instruction set architecture
o High-performance over military temperature range (200 nsec add; 1.8 μsec 16 x 16 multiply; 970 kips DAIS mix;* 1.5 mips DAIS mix)
o Single- and double-precision arithmetic (16 and 32 bits--16 general-purpose registers)
o 32- and 48-bit floating point arithmetic implemented on-chip
o Real-time processing with 16 levels of interrupt vectors, direct memory access, 128 input/output channels, and two programmable timers
o Directly addresses 64K words; extendable to 1M words with memory management unit
o Extensive fault detection and debugging capability with microcoded console support and self-test
o I^3L-II (3-micron) technology with 10^5 radiation tolerance
o Static operation with clock frequency 0-20 MHz
o Low-power Schottky input/output with multiprocessing capabilities
o Single 5V supply (additional injector current source required); power dissipation 2.5W
o Uses existing F9445 support circuits

*DAIS Mix is an Air Force specified set of instructions.

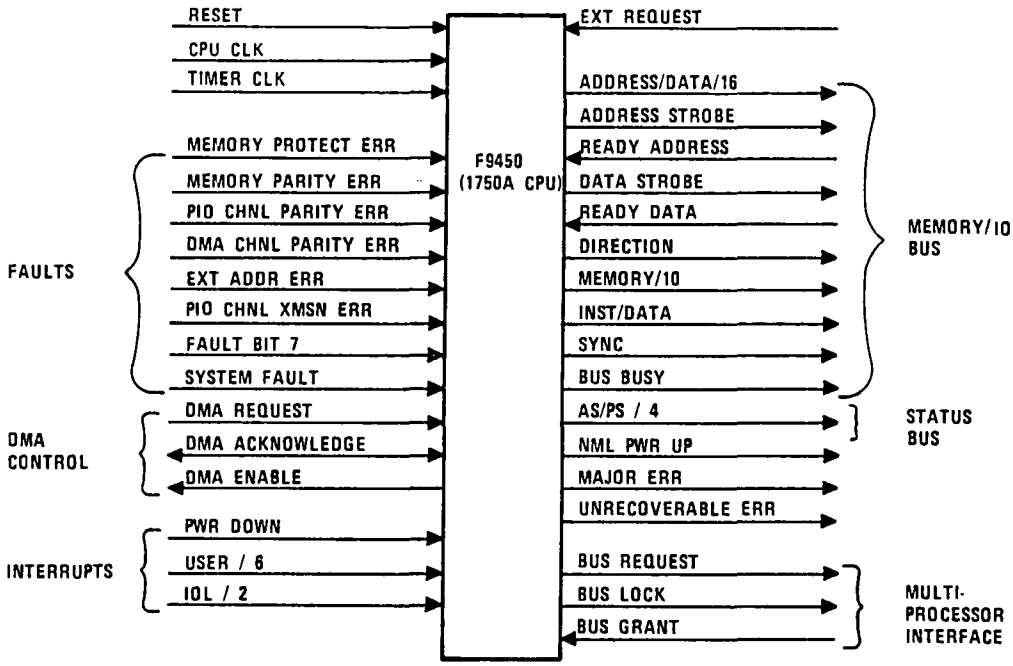


Figure 10. - F9450 logic symbol.

Trends in microprocessor throughput are shown in figure 11. The 1750A throughput for an Air Force specified instruction mix (including floating point) is 970 kops. This is comfortably in excess of flight control requirements (estimated as no more than 400 kops).

In addition to the Fairchild 1750A, other VHSIC programs are directed at the 1750A standard instruction set and will provide upgrades to our architecture.

Software development. - It is important to use a HOL for developing software. At present, the 1750A is supported by JOVIAL-J73. Our compiler was written in FORTRAN and developed by Software Engineering Associates under Air Force Avionics Laboratory Sponsorship. The compiler is presently hosted on IBM 360/370 computers.

JOVIAL is considered an interim language as the DoD makes the transition to Ada. Effective use of Ada requires an Ada Programming Support (APS) environment. Such tools are in development, but may require several years of

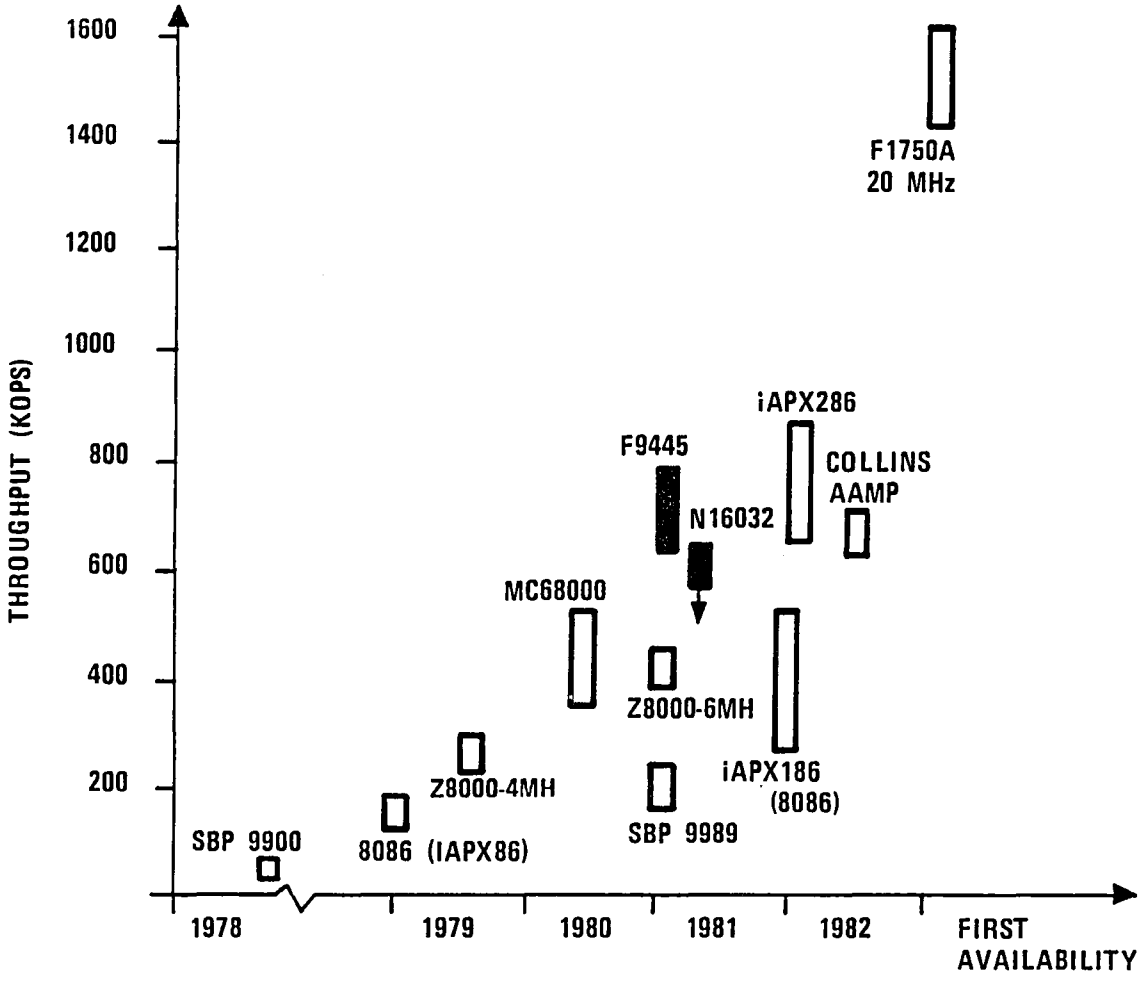


Figure 11. - Sixteen-bit microprocessor comparison.

effort. A hard look at using Ada as the HOL is recommended. It is felt that a significant contribution could be made by applying Ada to a real-time, ultra-reliable system.

2.3.3 Actuators

Triplex servo channels are employed for each control surface--rudder, aileron, and elevator. The servo interface uses a remote terminal, as shown previously in figure 3. The number of such remote servo terminals will depend on the particular application. For the S-3A, a mid-fuselage location for the ailerons and an aft terminal for the rudder and elevator seem appropriate.

The basic overall function of the remote terminal is to select a suitable signal from six or less computer channels and drive the surface accordingly. A detailed functional specification for the remote terminal is given in section 3.3 as an illustration of the design methodology. This paragraph summarizes those specifications. The terminal must receive and interpret serial digital transmissions from the computer channels. It must compute the control command for the electro-hydraulic servo valve (EHSV). This command must be sent to the servo amplifier and monitored. The remote terminal also must monitor each actuator channel.

This section presents some implementation details on a servo terminal that accomplishes these functions. The recommended implementation has three self-checking servo channels. Associated with each channel is self-testing health monitoring logic, which disengages the channel when a failure occurs. The health monitor sends back a discrete to the control panel for status monitoring. The three channels operate in an active/on-line mode. These features are discussed below.

One channel of basic servo electronics and actuation is shown in figure 12. Six serial buses from the self-checking computers are optically isolated and enter the multiplex chip. Any one of the six received digital commands can be converted and used. The particular input that is used depends on the address generated by the priority encoder. The selected input goes to the receiver (UART), the D/A, and the servo amp. The receiver contains a time-out feature such that if no message is received on the priority digital channel, a

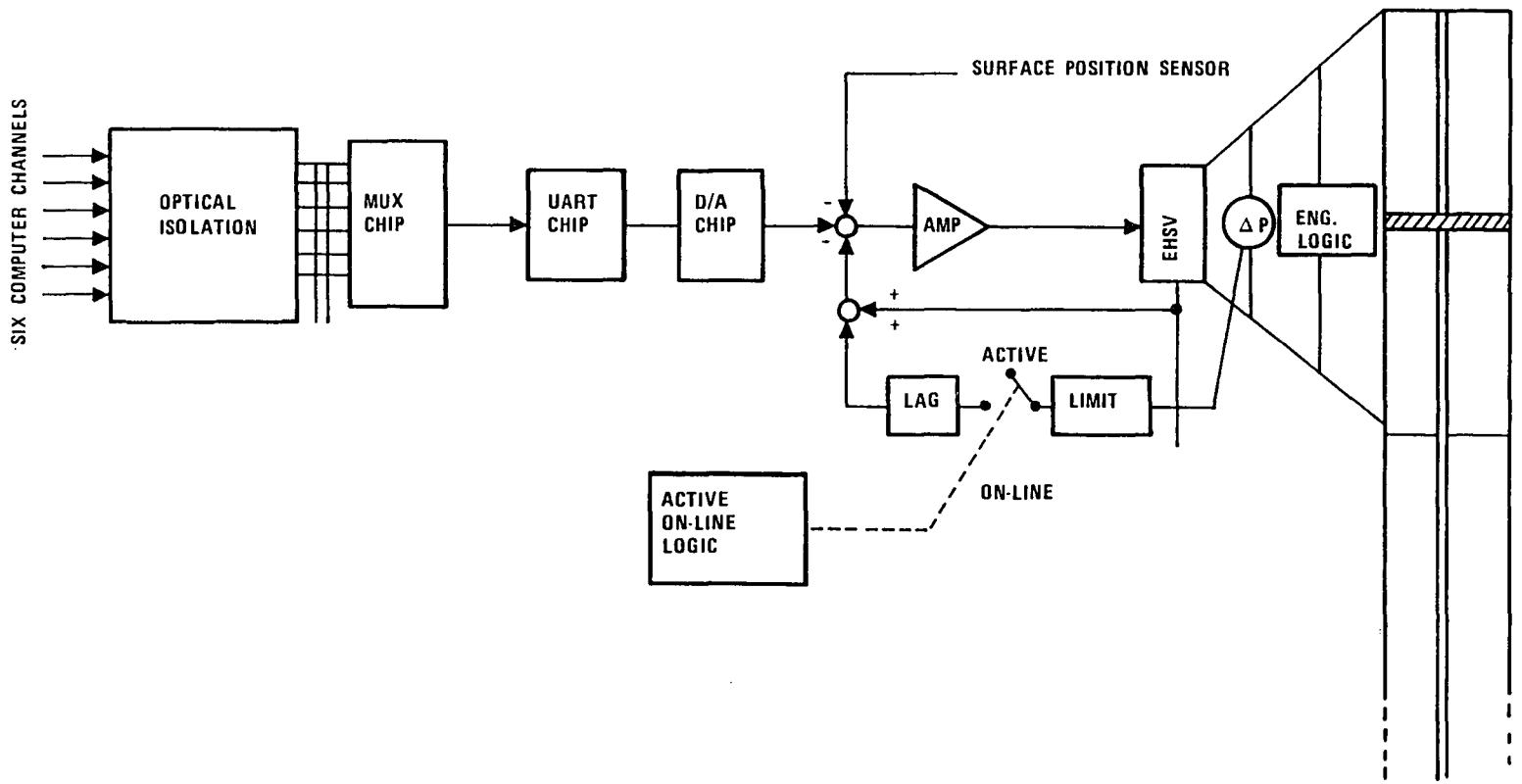


Figure 12. - One servo channel.

predetermined alternate is used. The servo amp provides an analog signal that positions the EHSV. This valve controls the flow of hydraulic fluid to position the cylinder connected to the surface.

Active/on-line control. - Any single servo channel is sufficient to position the surface. As shown in figure 12, all engaged cylinders are force-summed to drive the surface. Active/on-line control is a mechanism to relieve the force fight among the force-summed cylinders. One cylinder is in control of the surface position, or active, while the other engaged cylinders are on-line.

Ideally, the several actuation channels could operate in concert with one another in an each-channel-active configuration. However, because of the high-pressure gain characteristics of actuator valves, small tolerances in the actuator control loops would lead to significant force opposition between channels.

Maximum actuator capability force opposition could occur over a small deflection either side of the commanded position. No precise control capability exists while maximum force opposition is occurring. A hysteresis type of nonlinearity would be observed when the actuator command is cycled back and forth.

The pressure feedback path in the on-line channels must overcome this force-fight tendency by driving the on-line actuators towards a zero force output condition under normal operating conditions. A limiter is provided in the pressure feedback path, as shown in figure 12. This limiter is set at a value such that the pressure feedback signal can slightly exceed the maximum tolerance between channels. Based on prior studies, the maximum tolerance between channels of actuation was estimated to be 2.8 mA when $\pm$ 8 mA full-scale valves were considered. A pressure feedback limit equivalent to $\pm$4 mA was selected for the on-line actuator. The 4 mA limit would be exceeded for 1/2 percent of full travel errant motion, after which the on-line actuator channel will oppose the motion.

Thus the pressure feedback is effective in allowing the output actuators to operate in harmony with one another. However, the on-line actuator (or actuators) will oppose any active channel malfunction.

Upon detection of any failure in the active channel by the monitor, one of the on-line channels will be switched to the active status as the malfunctioning channel is bypassed.

Active or on-line status for all healthy servo channels is assigned on the basis of the channel health monitor signals. If a channel is bypassed, active and on-line have no meaning. Table 3 shows a logic table description of servo channel status as a function of failure status. This table reduces to the following simple boolean functions, which can be implemented in hardware:

A Active = $\overline{\text{A Eng}}$

B Active = $\overline{\text{A Eng}} \cdot \overline{\text{B Eng}}$

C Active = A Eng $\cdot$ B Eng $\cdot$ C Eng

Because of the limiting of the ΔP signal, assignment of active or on-line status is not critical. The "two active" and "none active" states are not flight-critical, so this logic is not required to be redundant. There is a slight preference to have failure modes set more than one channel active (force fight) in the event of a malfunction of the logic.

TABLE 3. - ACTIVE ON-LINE LOGIC

Condition	Servo Channel			Status		
	A	B	C	Active	On-Line	Bypassed
No Failures	OK	OK	OK	A	B,C	None
First Failures	F	OK	OK	B	C	A
	OK	F	OK	A	C	B
	OK	OK	F	A	B	C
Second Failures	OK	F	F	A	-	B,C
	F	OK	F	B	-	A,C
	F	F	OK	C	-	A,B

Self-checking health monitor. - The channel health monitor is responsible for detecting single errors in the channel and cutting off the channel (by the engage/bypass valve) if an error is detected. Once the engage/bypass valve is in the bypass position, it remains so unless there is a master system reset (probably a manual action). Therefore, it is sufficient for the channel health monitor to be able to detect only single failures in an unbypassed servo set. Multiple simultaneous failures are too improbable to consider. The bypass action clears each failure as it is detected.

It is essential that the channel health monitor be self-testing, though not necessarily failure-operational. This means that if there is a single fault in the channel health monitor logic, it is detected and forces the engage/bypass valve to the bypass position. Thus a servo channel is removed whenever a fault is detected in the channel or the channel health monitor. With three redundant channels, failures in up to two channels can be tolerated, thus satisfying the primary requirement of the servo actuator system.

The complete diagram of one servo channel is shown in figure 13. This figure adds the self-checking electronics and health monitor to the electronics and valves of figure 12. The path from the computer command to EHSV position is checked by a parallel path consisting of a redundant multiplexer and D/A chip and a model of the servo amp and valve. Note that the servo amp input includes a ΔP signal that does not appear as an input to the model since this signal is not involved in positioning the EHSV. The cylinder position, EHSV spool position, and ΔP measurements are made by LVDTs. These signals are monitored by validity discretes. If the measured EHSV position compares with the model prediction, and if the LVDTs are valid, the health monitor decides the servo channel is healthy and should be engaged. If the channel is not healthy, the monitor bypasses the channel. This is affected through a solenoid-held engage/bypass valve that controls the status of the cylinder.

An example design of the channel health monitor is shown in figure 14. The hardware is duplicated in one-out-of-two codes in order to detect all single faults and all multiple undirectional faults (e.g., due to power loss). The engage/bypass valve is engaged only when the two signals Channel OK and Channel Fail are respectively 1 and 0. In all other cases (11, 01, and

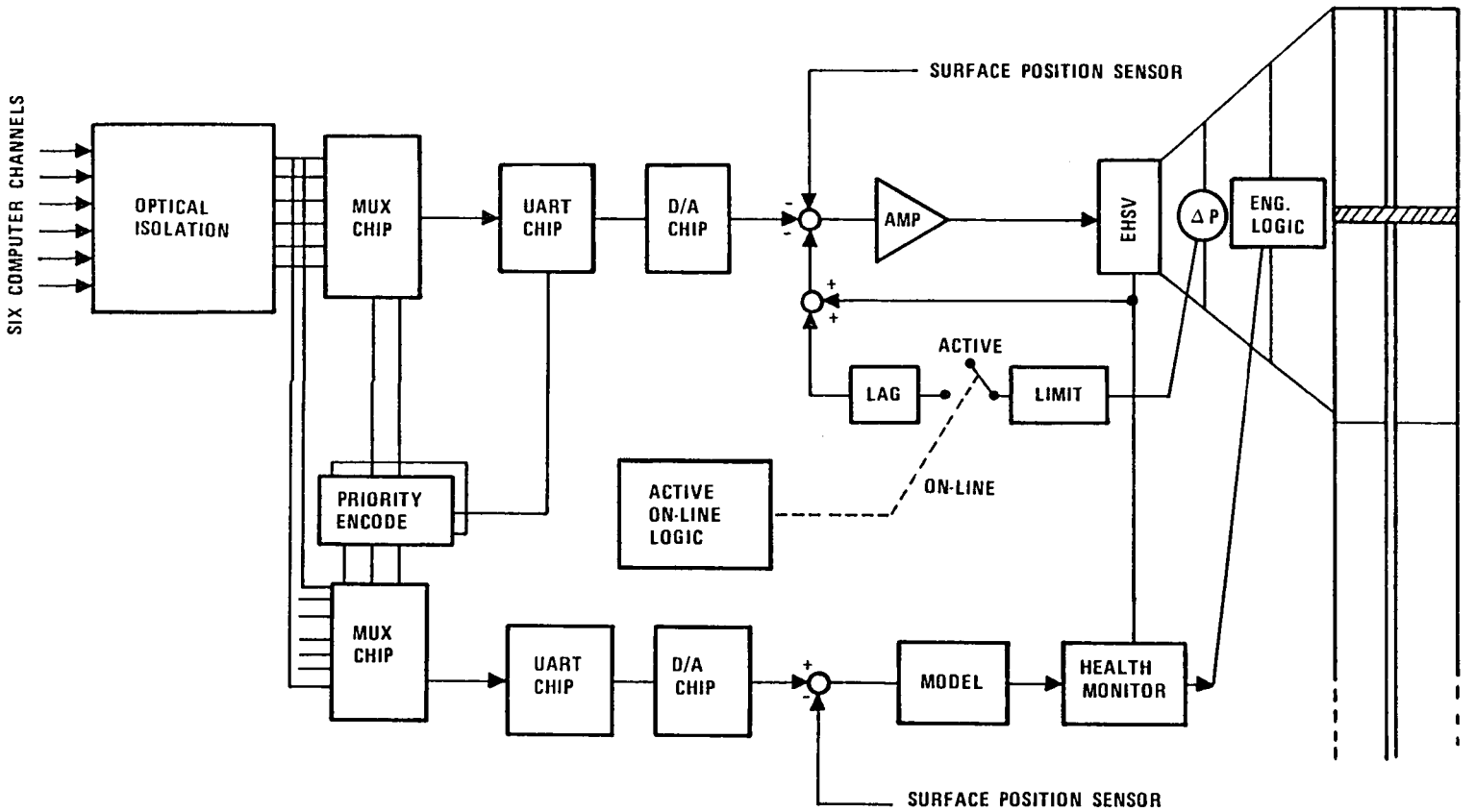


Figure 13. - One servo channel including self-diagnosis.

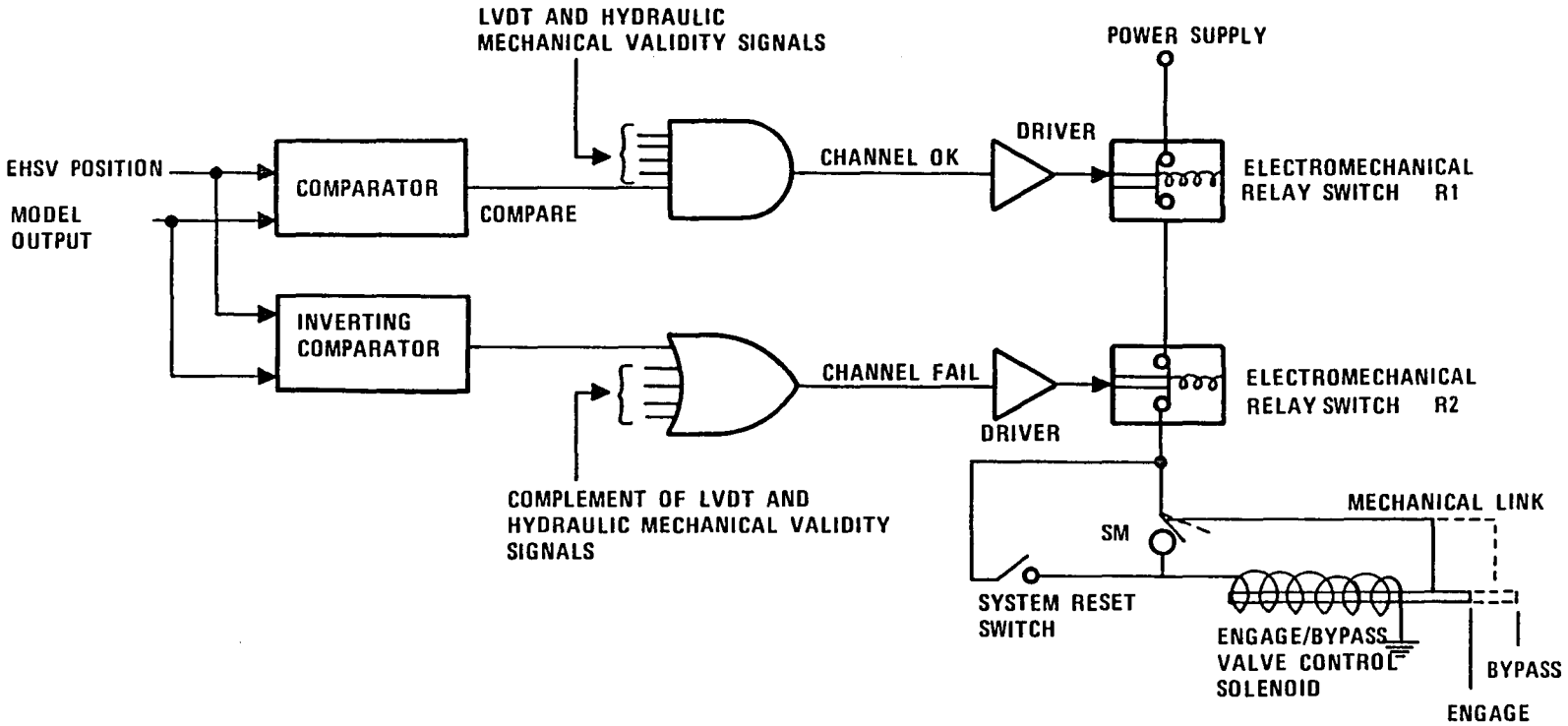


Figure 14. - Example of channel health monitor and engage/bypass valve control.

00) the valve should be disengaged. This logic function can be realized, as shown in figure 14, by putting two electromechanical relay switches in series. When the signals Channel OK and Channel Fail are respectively logic 1 and 0, both the switches are closed and current flows in the solenoid, keeping the engage/bypass valve in the engage position. For all other values of these signals, at least one switch will be open, causing the valve to go to the bypass position. The switch SM is a mechanical switch directly controlled by the position of the engage/bypass valve. It is open when the valve is in bypass position.

This example is presented to illustrate that self-testing mechanisms can be realized using simple coding techniques such as the one-out-of-two code. The necessary logic and coding functions can be achieved using any appropriate combination of digital electronics, analog electronics, electromechanical hardware, or mechanical hardware. Several design solutions exist. The actual solution to be used will depend on the detailed design tradeoffs, reliability of components, and requirements/constraints of a particular system. Using a coding technique allows the design of mechanisms that can test single failures, multiple undirectional failures, loss of power, open circuits, etc. Thus, the channel monitor/engage circuit can be implemented as a fault-secure circuit with the preferred failure state being a bypassed channel.

This completes the description of the recommended implementation. This implementation is compatible with use of the S-3A as a testbed and is based on components that could be available for flight test in the mid-1980's. The architecture could be implemented with other components and can assimilate upgraded processors, new sensors, or electromechanical actuators. A methodology for validating this architecture is detailed in the following sections.

SECTION 3--SYSTEM SPECIFICATION AND VALIDATION

Methodologies for the design and verification of general software systems have received much attention (ref. 9). There are heavy economic incentives

for reliability and maintainability. For flight controls, safety is the issue. The spectre of a single software error shutting down a multichannel redundant system haunts the avionics industry. We claim that this chimera should be banished for current synchronized-channel architectures (ref. 10) and certainly for the self-checking channel architecture described in the previous section. This confidence is not gained without cost. The system and its software must be designed to have simple, auditable, and testable structures. The reviews and tests must be done with a discipline that only the ultimate flight test can inspire. This section and sections 4 through 6 present a methodology for achieving and demonstrating this level of performance.

This section discusses the system specification and the definition of the hardware/software interface. Our methodology suggests an approach to making the system specification precise and complete (ref. 11), then adds more detail to describe the hardware/software interface (ref. 12). The ADFBW remote terminal is used as an example to conclude this section.

Section 4 addresses system reliability. Techniques for predicting reliability are recommended. The fault tree approach described is applied to the ADFBW architecture defined in section 2. Laboratory methods for validating reliability predictions conclude section 4.

Section 5 is concerned with software design and validation. It is noted that software is a paper business--a structure must be imposed. Suggestions for preparing the software specifications and designing, coding, and testing the software are given.

Section 6 considers the analysis and testing of the integrated system. Techniques for identifying test cases are presented. It is shown that much of this work can be automated to provide rapid and complete reviews of the performance and establish the correctness of the hardware and software.

3.1 METHODOLOGY OVERVIEW

An overview of the methodology proposed for developing the ADFBW system is shown in figure 15. This methodology is not novel; much of it is current

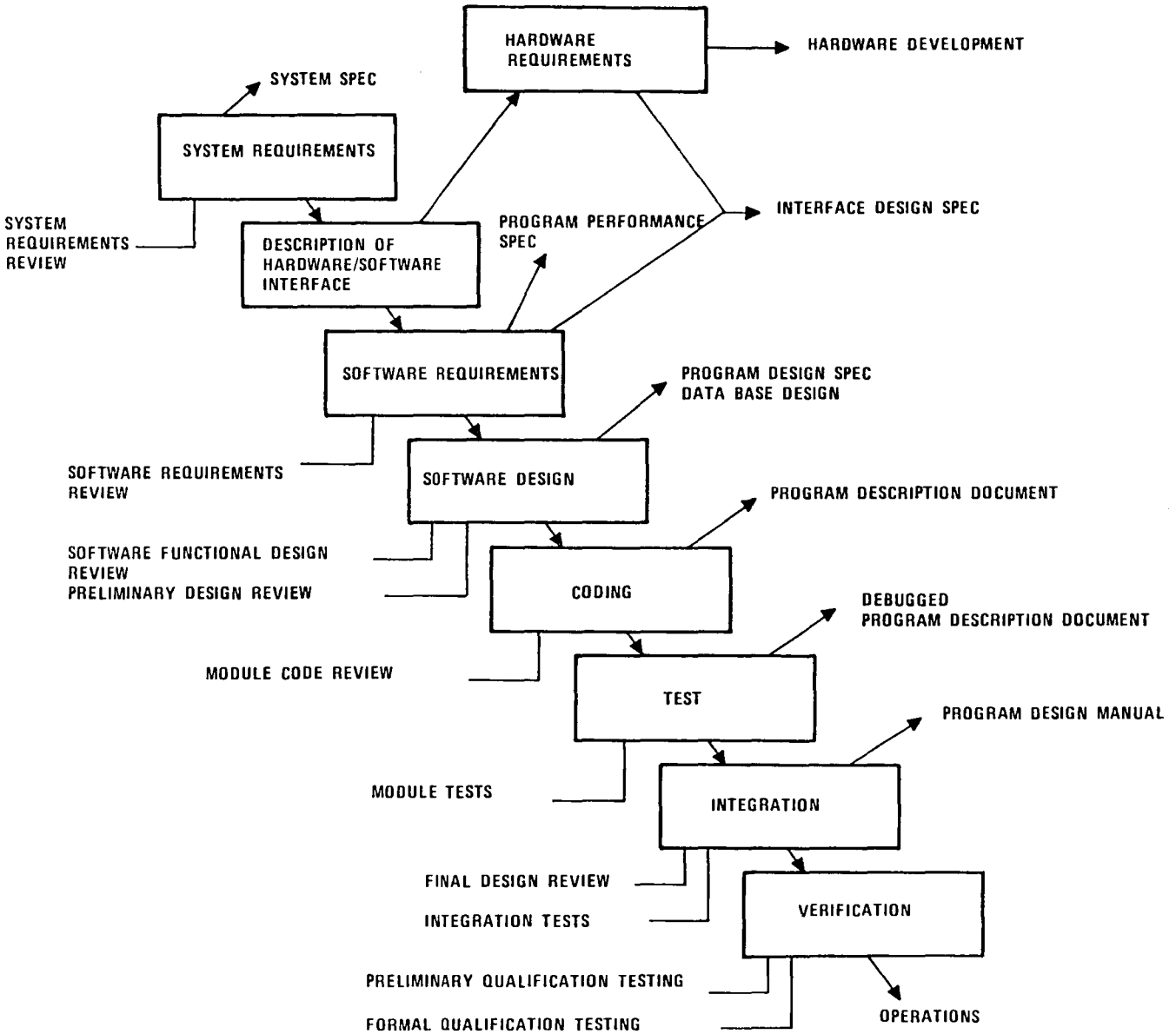


Figure 15. - Methodology overview.

practice. The methodology provides two key attributes of a successful development cycle:

(1) The cycle has a series of definable phases

(2) Each phase has unique, measurable outputs

The methodology presented in figure 15 requires the following activities:

System requirements review

Software requirements review

Preliminary design review

Software functional design review

Module code review

Module tests

Integration tests

Preliminary qualification tests

Formal qualification tests

These activities audit the transmission of the system requirements from the customer to the contractor, the translation of the customer requirements into system specifications, the extraction of software specifications into functional modules, the definition of hardware/software interfaces, and the further elaboration of detailed software design and testing. The first three phases relate to establishing the system requirements, hardware/software interface, and software requirements. They are outlined below. The remaining items, concerning the design and test of the software, are the subject of section 5.

Figure 15 shows typical documents produced in each phase. Most of these documents are used in the next step of the cycle. In addition, each phase requires some support functions (reviews) that need attention. Finally, various feedback paths (not shown) exist at each phase to resolve problems. The reviews at each step of the process are intended to remove errors at the most cost-effective level.

Since there is much experience in the design and coding of flight control software, there is confidence that the test and reviews will detect any errors.

3.1.1 System Requirements

This phase is conducted to determine the degree of completion of the concept definition, to review changes authorized by the customer, and to provide the details and background for preparing the system specifications. The requirements review provides an opportunity to impress on the customer that changes are expensive and that they should not be requested casually or capriciously.

3.1.2 Description of the Hardware/Software Interface

We recommend that the system be specified to the detail of identifying states, transitions, and inputs/outputs for all flight control functions abstractly without regard to hardware or software mechanizations. Of course, the allocation of functions between hardware and software is largely determined for the particular architecture, and from experience with similar systems. The object here is to add details and to make the description of the hardware/software interface complete and precise. After the interface has been defined, the software specifications are written. Sometimes a formal hardware specification is also prepared. The verification step must show that a system operating according to the hardware and software descriptions fulfills the original system specifications.

3.1.3 Software Requirements

The software specifications are written from the system specifications and the allocation of functions to hardware or software implementation. This defines the hardware/software interface. The software requirements review is informal; it will be conducted early in the program to ensure that the requirements are complete, necessary, and consistent. An informal demonstration will be made to show that the software as specified, plus the hardware functions, will fulfill the system specifications.

To summarize, it is our position that the knowledge and techniques exist to produce a validatable flight control system. The process requires a commitment of resources and the discipline to succeed. There are many historical differences in hardware and software development practices, as illustrated in table 4. Our proposed methodology imposes the same level of

TABLE 4. - HISTORICAL DIFFERENCES IN HARDWARE AND SOFTWARE DEVELOPMENT PRACTICES

o Hardware proofs requirements with breadboards
--Software typically attempts one continuous development
o Hardware requirements "freeze" prior to build
--Software often tolerates requirements changes throughout design, coding, and checkout
o Hardware makes design review compulsory
--Software has loosely defined design review points
o Hardware uses firm test procedures
--Software often debugs by engineering judgment
o Hardware uses standard parts
--Software is largely new sequences of computer instructions
o Hardware is built from prints
--Software generally has less detail provided at the design stage
-- The programmer has near-infinite variation available in implementing the design

structure and rigor on software as has evolved in the hardware area. This is the key to successfully developing flight-critical computer systems.

3.2 SYSTEM SPECIFICATIONS

Many studies have shown that precise and complete specifications return, many times over, the investment in their preparation. The terms "requirements" and "specifications" are not completely defined. We generally use requirements to mean the informal statements about the functions and performance of a system. These are prepared by the customer and are often not precise or complete. The specifications, or requirements specification, are the documents that try to outline the requirements in a more formal manner. These must be as complete and precise as needed to ensure the success of a project. Indeed, many errors are made in obtaining the correct description of what a system is supposed to do. The final validation of a system returns to these specifications.

In converting from requirements to specifications, there are varying degrees of formality. These range from formal languages like SRI's SPECIAL (ref. 13) to documents prepared according to various military standards (ref. 14).

3.2.1 Current Specifications

Many engineering projects use one of the military standards as a guide in writing specifications. The major shortcomings in following military standards are that some items may be omitted. There are no procedures to enforce completeness. In addition, the description of functions is left open. In this section specifications following NRL guidelines (ref. 15 and 16) and MIL-STD-483 are reviewed. Recommendations are made for specifying the various flight control functions. Finite-state machines are shown to be useful for describing mode switching, signal selection, and failure management functions.

To understand the limitations MIL-STD-483 imposes, the specifications for the NASA Demonstration Advanced Avionics System (DAAS) flight controls were written in the two styles (ref. 16) of MIL-STD-483 and the Naval Research Laboratory (ref. 17). The NRL style included the use of finite-state machines to specify the control modes.

The organization of the NRL specification is shown in table 5. The NRL suggests that we should:

(1) Specify only external behavior without implying a particular implementation.

(2) Specify constraints on the implementation, especially the details of the hardware interfaces.

(3) Write the document so that it is easy to change and may be kept current throughout the life cycle of the system; also so that it will serve as a reference to answer specific questions quickly, rather than to explain in general what the program does.

(4) Record forethought about the life cycle of the system, particularly to anticipate and facilitate later changes.

(5) Characterize acceptable responses to undesired events and not leave this to invention by the programmer.

TABLE 5. - ORGANIZATION OF AN NRL SPECIFICATION

0	Introduction	Organization principles, abstracts for other sections, notation guide
1	Computer Characteristics	If the computer is predetermined, a general description with particular attention to its idiosynchrasies; otherwise, a summary of its required characteristics
2	Hardware Interfaces	Concise description of information received or transmitted by the computer
3	Software Functions	What the software must do to meet its requirements, in various situations and in response to various events
4	Timing Constraints	How often and how fast each function must be performed; this section is separate from section 3 because "what" and "when" can change independently
5	Accuracy Constraints	How close output values must be to ideal values to be acceptable
6	Response to Undesired Events	What the software must do if sensors go down, the pilot keys in invalid data, etc
7	Subsets	What the program should do if it cannot do everything
8	Fundamental Assumptions	The characteristics of the program that will stay the same, no matter what changes are made
9	Changes	The types of changes that have been made or are expected
10	Glossary	All documentation is fraught with acronyms and technical terms. At first we prepared this guide for ourselves; as we learned the language, we retained it for newcomers
11	Sources	Annotated list of documentation and personnel, indicating the types of questions each can answer

(6) Formulate questions before trying to answer them.

(7) Separate concerns so that the scope of changes is limited.

(8) Be as formal as possible by using precise and consistent notation.

MIL-STD-483 calls for two documents. In part I the requirements for design, development, functional performance, test, and qualification are given. In part II the details of the configuration and the program itself are recorded as the final documentation for the item.

From the application to the DAAS flight controls, the following conclusions were drawn. Both approaches place great emphasis on getting the hardware interfaces clearly announced. It was found that very minor items in the interface have profound implications for the structure of the system and the software. Both approaches require careful annunciation of inputs and outputs. The military standard requires this for each function; the NRL seems to tend toward this at the software systems level.

Generally, the military standard was found to be inflexible and awkward, while the NRL style had the flexibility to fit the application. Many of the requirements set down in the military standard were answered by pat formulas which had very little real content. The section on testing is usually written this way, with promises that have no substance. Until a definitive methodology for validation is worked out for flight controls, this will be the case.

The NRL specifications do not consider the testing problem. Motivation and general descriptions were harder to include in the NRL outline. It was found that when changes came through, it was easy to change the finite-state machine descriptions, but it was hard to keep the general descriptions consistent throughout the documentation.

The recommendation is to use the outline of MIL-STD-483, but to use the suggestions from NRL and others to make a more complete and meaningful document. The flight control functions should be described in a manner that best fits each particular function. Some alternate modes of description are reviewed below.

3.2.2 Functional Descriptions

Any information or signal processing system may be thought to be made up of two flows--one is the information or data being processed by the system, the other is the sequence of control actions that manipulates the data (ref. 17).

Petri nets (ref. 18) and LOGOS (ref. 19 and 20) are two graphical techniques for describing flows. A Petri net is a directed, bipartite graph of alternating vertices called places and transitions. It provides an abstract model of information and control flows. The major applications of Petri nets have been for systems of events in which some of the events occur concurrently, but with constraints on the concurrence, predecence, or frequency of the events. Petri nets are sketched as circles and bars, called places and transitions. The places represent states; the transitions are labeled with the events that enable the transition. The graphical technique LOGOS portrays these two flows in parallel graphs. The control graph initiates, sequences, and synchronizes the data operations on the data graph. LOGOS has been used to analyze very complicated systems, including the Air Force DAIS architecture (ref. 21).

In many systems the structure for producing one of the flows is more complicated or fundamental to the system than the other. For example, in handling huge quantities of data, the organization of the data is central in designing efficient algorithms. One might say in this case that the data flow dominates the design considerations. For flight controls, the calculations on the data are not complicated, but the structure for controlling the computations is. Control flow dominates. The design will then be concerned chiefly with the control structure; the data flow will follow along naturally.

Both LOGOS and Petri nets were found to be awkward, very complicated, and difficult to change or analyze without a great amount of effort. Finite-state machines or other direct descriptions appear to be appropriate for flight controls.

A finite-state machine is the simplest computing structure. At the next level are the push-down automata, which have stack memories. The most general theoretical computing structure is the Turing machine. Finite-state machines use two expressions, called states and events. The states correspond to the

sequential circuits of the electronics engineer. Events represent an input to the control structure, signalling some important point of activity in its environment.

The advantage of the finite-state machine representation is that it is precise and may be easily reviewed by control engineers for completeness. It may be used to describe system-level functions; it is not limited only to hardware or software. The states must be clearly identified and the events causing transitions must be defined. This provides a structure that may be completely tested. Fortunately, all flight control functions are either straight-line calculations requiring no past data, or calculations requiring only a fixed, finite set of past data. Hence, the latter functions may be represented as finite-state machines.

A general finite-state machine is diagrammed in figure 16. When inputs are received, outputs are calculated as functions of the current values of the state variables and the inputs. Then the machine switches to a new state, again as a function of the current state and the input quantities. It is often useful to produce outputs associated with these state transitions; for

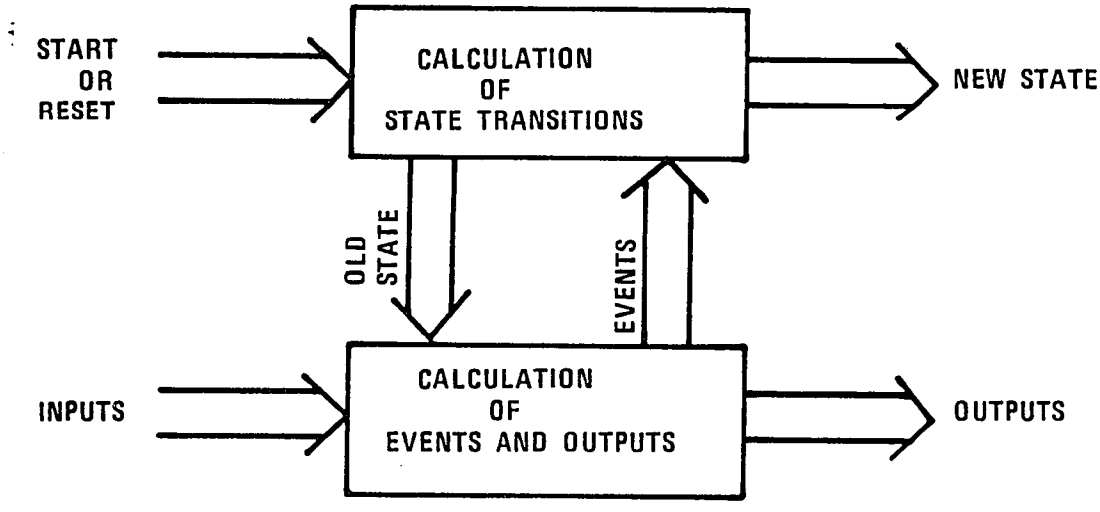


Figure 16. - A finite-state machine.

example, a warning to the pilot upon automatic change of mode with loss of an input signal. These representations were found to be very natural for mode switching, signal selection, synchronization, and failure management.

Finite-state machines are represented in two different ways--as a directed graph or a table. The directed graph approach is more intuitive because it is a picture. As the number of states, events, and state transitions grows, this advantage is effectively negated by the complexity of the diagrams. States are represented in the diagram as circles; legal state transitions are displayed as directed arrows connecting one state with another. The event that triggers a particular state transition is labeled on the arrow. The diagrams are interpreted in this way. At any time, the finite-state machine is in a current state. When an event is detected and received, the machine will make the state change indicated by the outgoing arrow labeled with that event. For a deterministic machine there can be at most one such arrow. When no such labeled arrow exists for the current state, this represents an error, and the machine will attempt to recover. The most simple recovery action is to ignore the event. The action sequence performed by the machine while changing state is generally not included in the diagram.

The alternative representation is to describe the state transitions with a table or matrix. The entries in the matrix contain the number of the new state and an ordered list of actions to be performed to effect a change in state (possibly null). Blank entries are illegal state transitions and could contain some code to assist recovery.

As the number of states and events grows larger, there is a need to partition the state machine so that each part is more manageable. To increase the clarity of the control structure, this partition should be done based on logical properties, and not in an arbitrary manner. The remote terminal function in section 3.3 illustrates how complex state machines can be partitioned into simpler ones.

It has been shown on the DAAS flight controls that describing mode logic as a finite-state machine is very effective. It makes all design decisions visible for review and helps prevent errors of omission. Signal selection algorithms may also be precisely described as finite-state machines. An example is given in appendix A. These representations were also used to

analyze synchronization schemes. This application is also recorded in the appendix. Part of a triply-redundant failure management segment for data exchange and voting is analyzed in the appendix by tracing the syndromes caused by component failures. However, while many of the functions in flight controls are finite-state machines, we can be flexible and use whatever is most appropriate.

The functions for flight control fall into categories for which general verification requirements will be prescribed:

(1) The executive structure (initialize, branch in the rate tree, recover from power interrupts, equalize integrations, maintain the dynamic filter states, annunciate system status)

(2) Data transfers (input, output, exchange data between channels)

(3) Control mode switching and dynamical switching within control modes

(4) Control law calculations (outer loops, inner loops, gain schedules)

(5) Synchronization (synchronize channels, time-synchronize programs for transfers, etc)

(6) Built-in-test functions (preflight checks, on-line checks)

(7) Selection from redundant input signals

(8) Failure detection and reconfiguration

It is also necessary to show a global consistency between these functions, particularly the built-in-tests and the failure management functions. For functions that have auxiliary hardware, as does the frame synchronization of channels, it must be shown that the response of the software to a hardware fault cannot result in a single-point failure.

3.3 FUNCTIONAL SPECIFICATION FOR THE REMOTE TERMINAL

The remote actuator terminal will be used as an example of writing functional specifications abstractly, independent of the implementation. The purpose of this approach is to have specifications against which the system can be validated, regardless of the details of the hardware/software allocations. This will also provide a measure for checking these allocations and for reviewing the hardware/software interface.

Implementation details of the remote terminal were discussed in section 2.3.3. (See the block diagram in figure 13.) This paragraph provides a brief summary of the implementation for convenient reference. The servo commands are serially transmitted to the remote terminal from six or fewer computing channels. These will carry parity bits with which the fidelity of the transmission and the status of the sending channel may be determined.

Three hydraulic cylinders are connected to sum forces to drive the aerodynamic surface and to provide triple redundancy. Any cylinder alone can position the surface. Each cylinder has a solenoid-held engage/bypass valve to control the status of the cylinder. The position of the cylinder, the position of the spool of the EHSV, and the pressure differential across the cylinder are measured by LVDTs. Each of these provides a signal attesting to the validity of the LVDT output.

3.3.1 Top-Level Function of the Remote Terminal

The remote terminal must select a suitable signal from six or fewer of the computer channels and drive the surface according to this command. The unit must be operational following any two component failures. The failure may be in mechanical, hydraulic, or electrical components. "Suitable signal" is not defined, but left as a choice in the design.

3.3.2 Hierarchy of Functions

The top-level function may be further specified in terms of the lower-level functions that are necessary. Figure 17 illustrates this decomposition. Design decisions are made in constructing this decomposition.

Drive EHSV according to command. - One of the second-level functions of the remote terminal is to drive the EHSV of each redundant channel. This is accomplished by obtaining the input servo command and computing the control valve for the EHSV. The serial digital transmissions from the six computing channels must be received and interpreted. The presence of a signal and the validity of the transmission must be determined by the subfunction "validate signal transmissions." The "select command" subfunction must choose from the valid signals or perform median, averaging, or some selection process. The

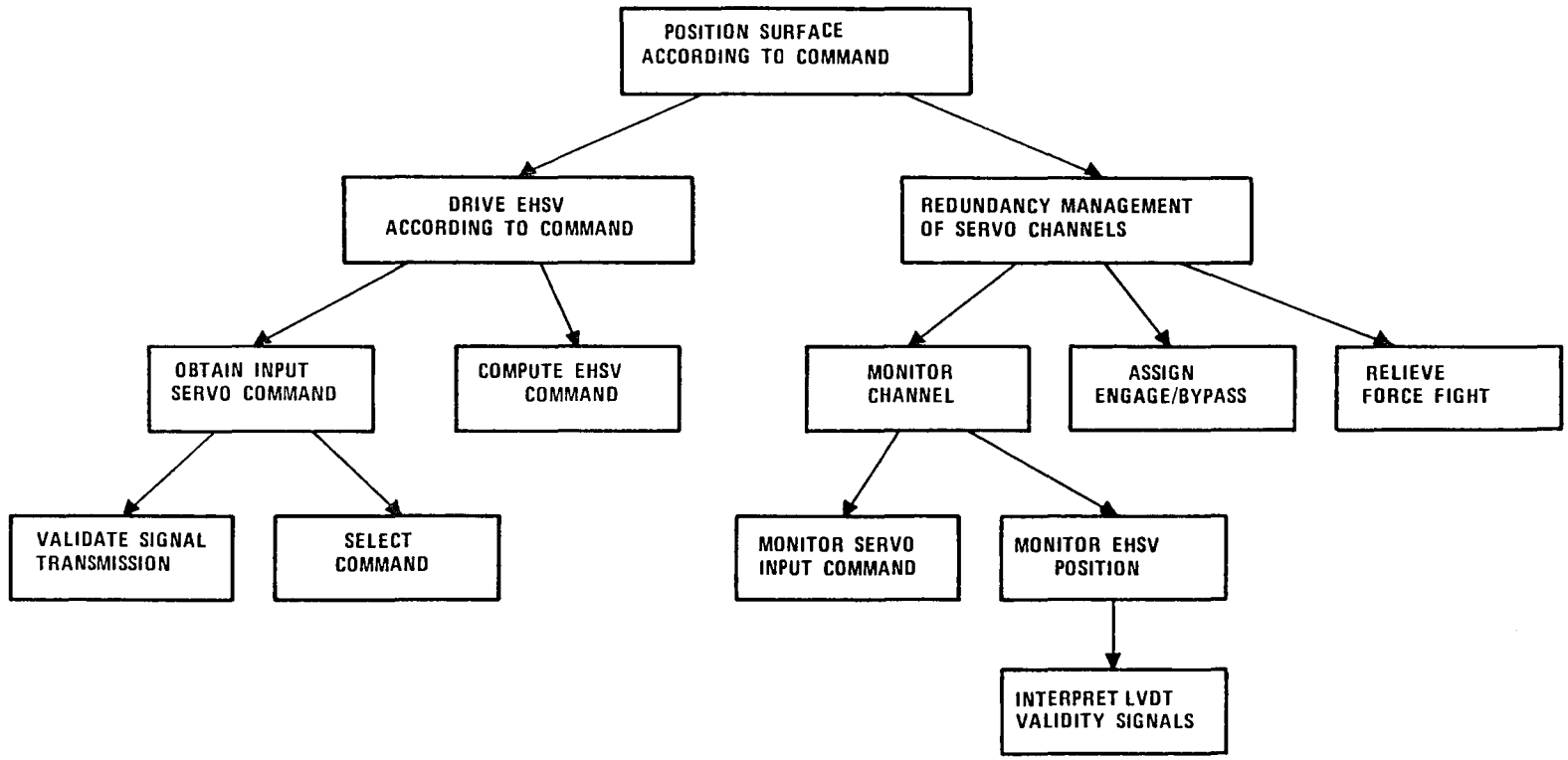


Figure 17. - Hierarchy chart of specifications for the remote terminal.

surface position, the valve spool position, and the differential of pressure in the cylinder are fed back and combined with the selected command as specified by the servo system control law. This control law computation may be analog or digital. An analog signal to drive the EHSV must be provided.

Redundancy management of servo channels. - The other second-level function of the remote terminal is to perform the redundancy management of the servo channels. Redundancy management includes (1) monitoring the health of each channel, (2) either engaging or bypassing a channel based on its health, and (3) relieving the force fight among the engaged channels. The monitor channel subfunction is to provide the failure detection mechanism for the general operation of the servo channel. It may require self-checking circuitry. Proper monitoring of any D/A or A/D translations is required. A comparison between expected valve-spool position as predicted by a model and the measured valve-spool position is suggested. A subfunction to interpret LVDT validity signals is required to determine that the feedback sensors are all functioning properly. Each channel must be engaged or bypassed on the basis of the output of the monitor channel subfunction. Also, a mechanism must be included to relieve the force fight among the force-summed cylinders. The use of an active/on-line assignment with pressure differential feedback is suggested.

3.3.3 Finite-State Machine Description

The redundancy management of the three servo channels can be specified as a finite-state machine. The health monitoring function of each channel is used to assign an engage or bypass status. An engaged channel may be active or on-line, as described in the previous paragraph and in section 2.3.3. Thus each servo channel can be in one of three possible states:

(1) Engaged and active

(2) Engaged and on-line

(3) Bypassed

There are $3^3 = 27$ states; the various transitions can be described based on changes in engage/bypass or active/on-line status. In order to study the engage/bypass function it is useful to cluster the 27 states into the eight groups shown in figure 18. Here transitions occur only if the engage

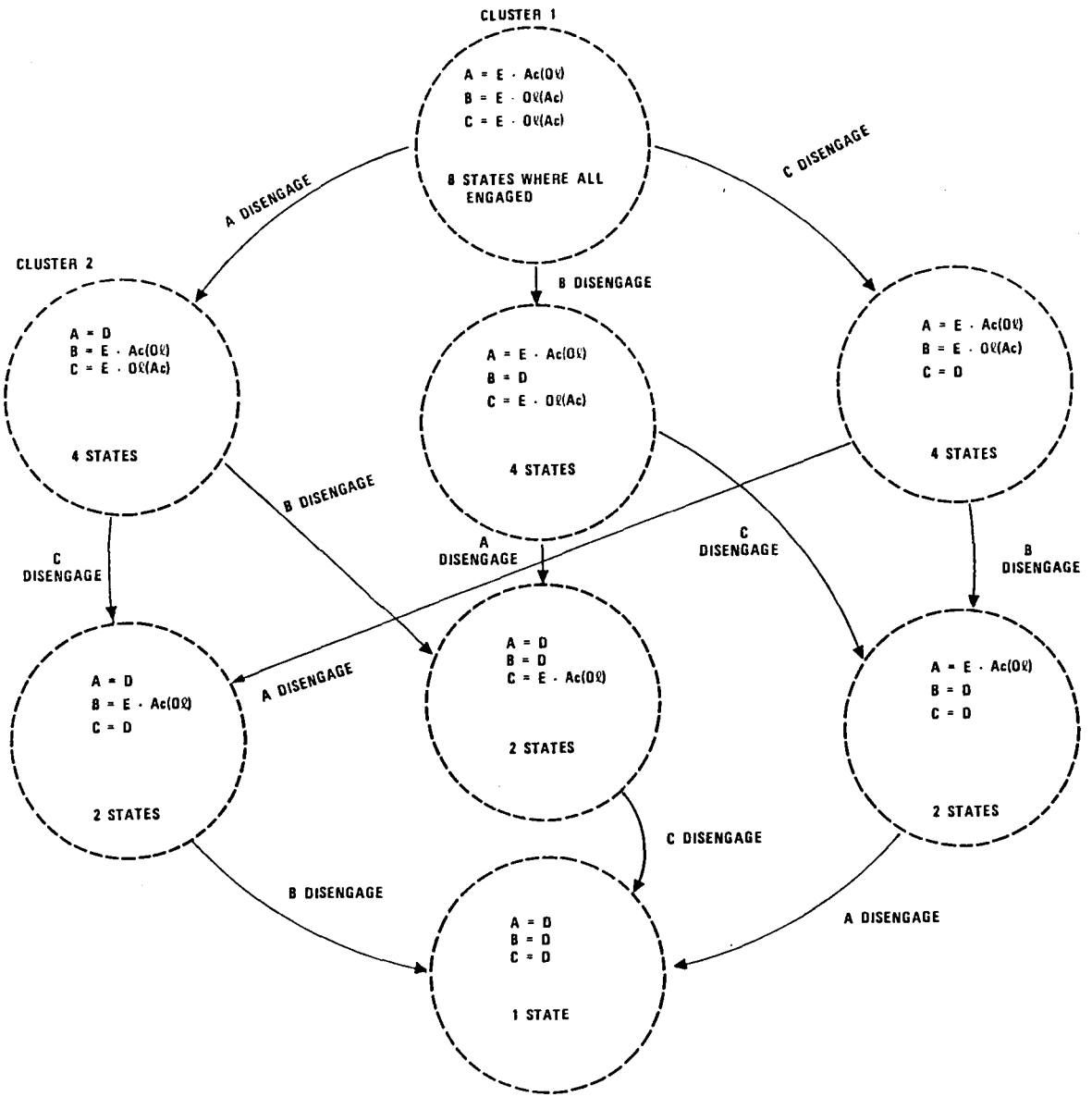


Figure 18. - Finite-state model for the remote terminal.

status of any channel changes. States within each cluster cover all the active/on-line assignments, including those resulting from logic failures (i.e., all active or all on-line). To verify this function in the remote terminal, it must be shown that all transitions between clusters operate as specified. For example, the event "channel A bypassed" must take any of the eight states in cluster 1 to one of the states in cluster 2. To examine the active/on-line logic, the states within each cluster must be delineated. The transitions between states within a cluster can be verified, and it must be shown that the active/on-line logic does not cause transitions between clusters.

The finite-state description of figure 18 was used as an example. It was also used to design the logic in the remote terminal presented in section 2.2.3. This description is useful because it requires the designer to consider all the possibilities. In addition, by clustering the states, the operation of the engage/bypass logic can be separated from the active/on-line function.

SECTION 4--SYSTEM RELIABILITY

A major aspect of the design methodology is the estimation and validation of the system reliability. Crude estimates of reliability are obtained during the process of defining the flight control architecture. Once a candidate architecture has been defined, a detailed analysis to estimate the probability of loss of control per hour of flight must be performed. This theoretical estimate of probability must be substantiated with laboratory tests. This section discusses both the calculation of theoretical reliability and laboratory methods to verify and validate the reliability of the system.

4.1 RELIABILITY ESTIMATION

Sound theoretical estimates of the performance of the ultra-reliable FBW system are necessary. In formulating the theoretical models for the reliability analysis, the following assumptions are made:

(1) All possible failure modes of the system are identified and their effects on the system's operation are known.

(2) The ability of the system to detect faults and to reconfigure automatically is implemented correctly by the software and the hardware of the system.

(3) All component failures are random in nature, and their failure rates are known.

(4) All possible interactions between the system and its environment have been foreseen.

These assumptions separate abstract concept from physical reality. It is difficult to identify all possible failure modes of a complex system and to foresee all possible interactions of the system with its environment. Only through years of experience with flight control systems has confidence been gained that the abstraction comes close to physical reality. It is only with these assumptions, however, that theoretical methods can be applied and the system's design can be based on numerical values such as 10^{-9} per hour probability of failure.

Before the detailed reliability analysis is performed, all pertinent component failure modes of the system are identified. Failure rates expressed as the MTBF of these components are estimated based on analysis, specifications, and experience with the actual unit, if it exists. The component reliabilities are combined in the statistical equations, from which the failure probability of the total system configuration can be computed. The equations take into consideration the number of redundant channels, the effects and interactions of a component failure on the other parts of the system, self-test coverage, and the reliability of each component.

4.1.1 Fault-Tree Analysis

Fault-tree analysis is the recommended method of computing statistical reliability. It provides a clear demonstration of the effects of system element faults, and computer tools aid its application. The fault-tree program used in this study can determine the sensitivity of system failure to the reliability of individual components. This serves to identify the critical components.

The fault tree graphically represents the logical relationship of a particular, undesirable event, called the top event, to the basic failures (causes) called primary events. If system failure is the undesirable event, then the fault tree would graphically represent all the possible faults or failures, or their combinations, that could cause the top event to occur.

After the failure model of the system has been expressed in the fault-tree format, the computer program, method of obtaining cut set (MOCUS), is used to provide qualitative analysis. MOCUS identifies and displays all critical failure paths (minimal cut sets) of a system's logic structure.

The output of MOCUS is then used as an input to the kinetic tree theory (KITT) computer program for quantitative analysis. KITT provides information on the probability of failure as a function of time for each component, for each minimal cut set, and for the entire system.

Before the algorithm is presented, some terms are defined as follows:

(1) Cut set. - This is a collection of basic events whose presence will cause the top event to occur.

(2) Minimal cut set. - A cut set is said to be minimal if it cannot be further minimized and still ensures the occurrence of the top event.

(3) Boolean indexed cut sets (BICS). - BICS are defined such that, if all the primary events are different, the BICS are precisely the minimal cut sets. This definition of BICS does not mean that the method is limited to fault trees with primary events appearing only once in the fault tree. The algorithm used in MOCUS starts with the top event and resolves the fault tree to obtain cut sets. An AND gate always increases the size of the cut set whereas an OR gate increases the number of cut sets. Cut sets obtained in this fashion are BICS. Duplicate events appearing in any one BICS, if any, are eliminated. All BICS that are supersets of any other BICS are discarded. This process is illustrated in an example below.

(4) Superset. - Superset is a BICS that contains every primary event that some other BICS contains, plus additional primary events. After this winnowing, the minimal cut sets are determined.

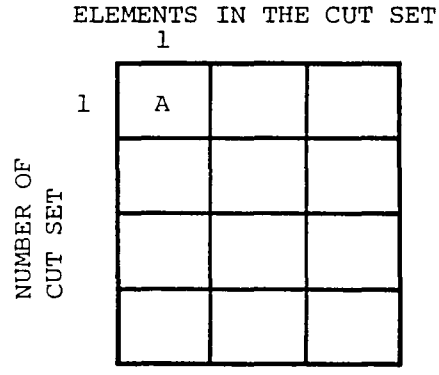
MOCUS. - The algorithm used in MOCUS to find the minimal cut set is unique in that it starts at the main event (failure) of interest, called the

top event, and proceeds to basic primary events (component failure) to resolve the fault tree into cut sets. MOCUS guarantees that all the minimal cut sets are found. A main feature of MOCUS is the small execution time it takes to determine all minimal cut sets even for a large, complex tree.

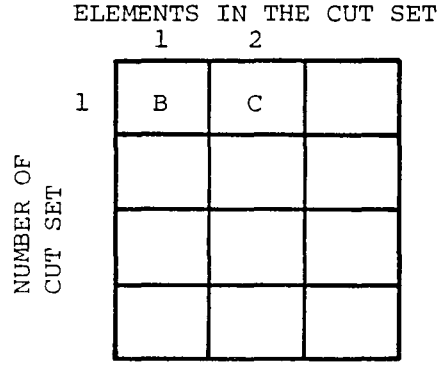
The MOCUS procedure uses boolean logic to determine the minimum group of cut sets that must be considered in determining the system reliability. This "minimization" greatly reduces the effort in calculating the reliability figures because it eliminates all duplication. An example of such minimization is illustrated below.

The construction of the minimal cut sets will be illustrated with a simple example. Consider the logic diagram shown in figure 19.

The algorithm starts with the top event. The gate under the top event is A.



The inputs to the AND gate A are gates B and C.



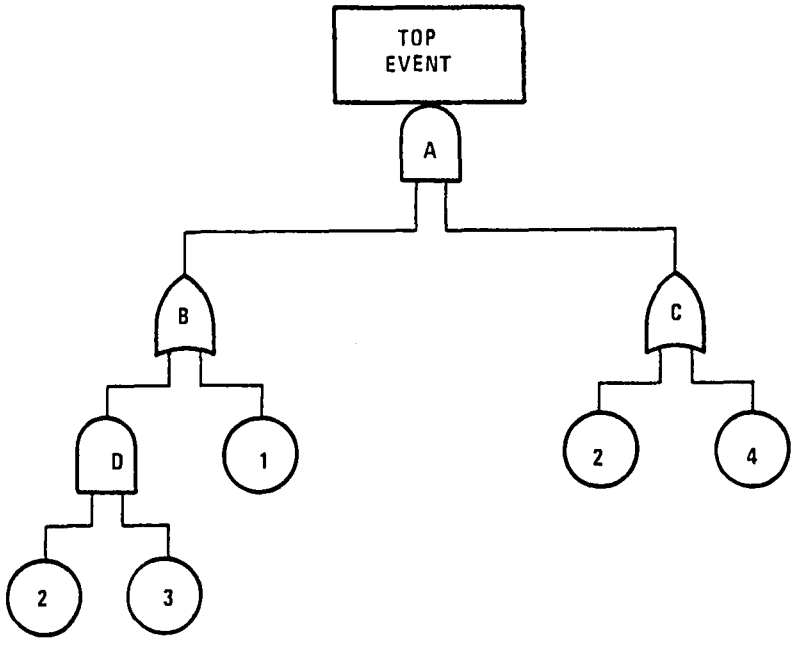


Figure 19. - Sample fault tree.

Gate A is replaced with its input, gates B and C. Gate B is an OR gate and its inputs are gate D and component 1. The OR gate increases the number of cut sets.

ELEMENTS IN THE CUT SET

NUMBER OF CUT SET	1	2	
1	D	C	
2	1	C	

Gate D is an AND gate with input of components 2 and 3. The AND gate increases the size (number of elements) of the cut set.

NUMBER OF CUT SET	ELEMENTS IN THE CUT SET 1	2	3
1	2	3	C
2	1	C	

Gate C is an OR gate with input of components 2 and 4. Replacing gate C, we have:

NUMBER OF CUT SET	ELEMENTS IN THE CUT SET 1	2	3
1	2	3	2
2	2	3	4
3	1	2	
4	1	4	

Cut sets number 1, 2, 3, and 4 are BICS. In cut set number 1, the basic event of component 2 failure is duplicated, thus eliminated.

NUMBER OF CUT SET	ELEMENTS IN THE CUT SET 1	2	3
1	2	3	
2	2	3	4
3	1	2	
4	1	4	

Cut set number 2 is a superset of cut set number 1, therefore it is discarded.

NUMBER OF CUT SET	ELEMENTS IN THE CUT SET 1	2	3
1	2	3	
2	1	2	
3	1	4	

For the sample fault tree the minimal cut sets are (2,3), (1,2), and (1,4). This result is used as input to KITT.

KITT. - KITT is a computer code written as an application of kinetic tree theory. KITT requires as input the unique minimal cut sets of the fault tree. Exact, time-dependent reliability information is determined for each component of the fault tree and for each minimal cut set.

The failure intensity (Δ) of each component is assumed to be constant with respect to time (i.e., exponential failure distributions only are considered). As in kinetic tree theory, the components are assumed independent. All the components are assumed to be in their operating state at t = 0. KITT can handle components that are nonrepairable or that have a constant repair time.

The system reliability information, or system reliability characteristics, obtained from the minimal cut sets is:

$Q_0(t)$ System failed probability--the probability that the system is in its failed state at time t.

$W_0(t)$ System failure rate--the expected number of failures the system will suffer per unit time at time t.

$\lambda_0(t)$ System failure intensity--the probability that the system will suffer a failure per unit time at time t, given it is in its functioning state at time t.

$_0\int^t W_0(t')dt'$ The expected number of failures the system will suffer during the time interval from 0 to t.

$1 - EXP[-_0\int^t \lambda_0(t')dt']$ The probability that the system will suffer one or more failures in the time interval from 0 to t.

Reliability results are obtained by upper-bound approximations. The upper bounds can be obtained when minimal cut sets are used to determine system reliability information. The upper bounds are excellent approximations to the exact values.

The system failure rate $W_0(t)$, failure intensity $\lambda_0(t)$, and failed probability $Q_0(t)$ are bounded and approximated as:

$$Q_0(t) \leq 1 - \prod_{i=1}^{NC} [1_i - Q^v(t)]$$

$$W_0(t) \leq \sum_{i=1}^{NC} W_i(t)$$

$$\lambda_0(t) < \frac{\sum_{i=2}^{NC} W(t)_i^{\cdot}}{\prod_{i=1}^{NC} [1 - Q^{v}(t)_i]}$$

where

NC = total number of minimal cut sets

$W_i^{v}(t)$ = i^{th} minimal cut set failure rate

$Q_i^{v}(t)$ = i^{th} minimal cut set failed probability

Using these upper bounds for $\lambda_0(t)$ and $W_0(t)$, upper bounds for the integral system characteristics can also be approximated as:

$$\int_0^t W_0(t')dt' \leq \int_0^t \sum_{i=1}^{NC} W_i^{v}(t') \, dt'$$

$$1 - EXP[-\int_0^t \lambda_0(t')dt'] \leq 1 - EXP\left[-\int_0^t \frac{\sum_{i=1}^{NC} W_i^{v}(t')}{\prod_{i=1}^{NC} (1 - Q_i^{v}(t'))} \, dt'\right]$$

The upper bounds become the exact values if the minimum cut sets have no components in common--that is, the minimal cut sets are independent. This is quite important since the values are thus always conservative, resulting in a conservative estimate for system failure phenomena.

4.1.2 Analysis of the Advanced Fly-by-Wire System

The flight control system can basically be divided into the control system architect and the S-3A interfaces. The proposed microprocessor control system architecture was given in section 2.

The S-3A interfaces are hydraulic and electrical. The baseline S-3A hydraulic system is a dual system. The power source for the hydraulic system is the two-engine-driven pump with an electric-driven pump as backup for system 1. The baseline S-3A electrical system consists of two engine-driven 75 kva generators and an APU-driven 5 kva generator. Backup batteries are provided for the FBW flight control computers. It is assumed that three backup batteries are operative at dispatch.

Three proposed refinements of the baseline configuration were studied. The goal was to improve overall system reliability.

The first refinement is to incorporate a bigger auxiliary power unit (APU). This provides the capability of driving a 75 kva generator in lieu of the 5 kva generator. With this additional electrical power, a bigger backup electric-driven hydraulic pump can be incorporated. The additional electrical power also enhances the electrical power system.

Another proposed refinement is to have three separate and independent hydraulic systems, with the additional third system powered by an electric-driven pump, without the benefit of the bigger APU.

The last proposed refinement is to combine the above refinements, having a bigger APU and three independent hydraulic systems.

The four configurations above were combined with the ADFBW architecture to construct fault trees for the following four cases:

(1) FBW 3

(2) FBW 3 with bigger APU

(3) FBW 3 with three hydraulic systems

(4) FBW 3 with bigger APU and three hydraulic systems

Throughout this analysis, the fault trees constructed are of the primary-failure and command-failure types. Secondary failure fault trees are not included in this phase of the study.

A component failure is considered a primary failure if it occurs while the part is functioning within the operating parameters for which it was designed. Command failures are failures of coordinating events between various levels of the fault tree, from basic failure events to the top event. Secondary failures are due to excessive environmental or operational stress placed on the system components.

Failure rates used in analyzing the S-3A equipments are point estimates of the in-service experience covering the 1980 and 1981 period. Estimates of the component failure rate were made for the new equipment.

A total of 62 components was evaluated in the fault tree analysis. The components are included in the basic FBW system architecture and the electrical and hydraulic systems of the S-3A. Only the dispatch model of the FBW architecture was used. The dispatch model is the FBW system configuration with a minimum number of operative elements allowed for dispatch. The components used in the analysis and their corresponding failure rates are summarized in table 6.

The loss of flight control probability for an hour of flight was evaluated for each of the four configurations as follows:

(1) FBW 3

$$P_F = 6.188 \times 10^{-8}$$

(2) FBW 3 with bigger APU

$$P_F = 1.111 \times 10^{-10}$$

(3) FBW 3 with three hydraulic systems

$$P_F = 6.227 \times 10^{-8}$$

(4) FBW 3 with bigger APU and three hydraulic systems

$$P_F = 1.843 \times 10^{-10}$$

A computer printout provided documentation for each of the configurations and failure rates used. It also displays all minimal cut sets and their associated failure probability, in descending order. If further reliability improvement is deemed necessary, attention should be focused on the components in the top-ranked minimal cut sets.

Configuration (2) above exhibits the highest reliability among the four. This configuration, in effect, has three hydraulic pumps with crossfeed capability between pumps 2 and 3. The disadvantage of this arrangement is that both systems are lost with the occurrence of a single failure, which depletes the hydraulic fluid.

The anticipated increase of reliability with three independent hydraulic systems--configuration (3)--was not observed. This was mainly due to the fact that the third hydraulic system was constructed from the backup pump of system

TABLE 6. - ADFBW FAULT TREE PRIMITIVES

No.	Computer Symbol	Failure Rate (10^{-6}/hour)	Description
1	AAPU	438	Accumulator for APU starter
2	AC1	91	Air data computer, channel 1
3	AC2	91	Air data computer, channel 2
4	AC3	91	Air data computer, channel 3
5	AC4	91	Air data computer, channel 4
6	APU	480	Auxiliary power unit
7	AX1	30	Longitudinal accelerometer, channel 1
8	AX2	30	Longitudinal accelerometer, channel 2
9	AX3	30	Longitudinal accelerometer, channel 3
10	AX4	30	Longitudinal accelerometer, channel 4
11	AY1	30	Lateral accelerometer, channel 1
12	AY2	30	Lateral accelerometer, channel 2
13	AY3	30	Lateral accelerometer, channel 3
14	AY4	30	Lateral accelerometer, channel 4
15	AZ1	30	Vertical accelerometer, channel 1
16	AZ2	30	Vertical accelerometer, channel 2
17	AZ3	30	Vertical accelerometer, channel 3
18	AZ4	30	Vertical accelerometer, channel 4
19	BPMP	38.7	Backup pump for hydraulic system no. 1
20	BT1	348.9	Battery no. 1
21	BT2	348.9	Battery no. 2
22	CC1	250.0	FBW computer, channel 1
23	CC2	250.0	FBW computer, channel 2
24	CC3	250.0	FBW computer, channel 3
25	CT1	40.0	Copilot stick transducer, channel 1
26	CT2	40.0	Copilot stick transducer, channel 2
27	CT3	40.0	Copilot stick transducer, channel 3
28	ENG1	247.0	Engine no. 1
29	ENG2	247.0	Egine no. 2
30	GTEN	114.0	APU electrical generator
31	IDG1	5000.0	Integrated drive generator no. 1

TABLE 6. - Concluded

No.	Computer Symbol	Failure Rate (10^{-6}/hour)	Description
32	IDG2	5000.0	Integrated drive generator no. 2
33	PG1	30.0	Pitch rate gyro, channel 1
34	PG2	30.0	Pitch rate gyro, channel 2
35	PG3	30.0	Pitch rate gyro, channel 3
36	PG4	30.0	Pitch rate gyro, channel 4
37	PMP1	209.0	Engine-driven hydraulic pump no. 1
38	PMP2	209.0	Engine-driven hydraulic pump no. 2
39	PS1	90.0	Elevator secondary actuator, channel 1
40	PS2	90.0	Elevator secondary actuator, channel 2
41	PS3	90.0	Elevator secondary actuator, channel 3
42	PT1	40.0	Pilot stick transducer, channel 1
43	PT2	40.0	Pilot stick transducer, channel 2
44	PT3	40.0	Pilot stick transducer, channel 3
45	RG1	30.0	Roll rate gyro, channel 1
46	RG2	30.0	Roll rate gyro, channel 2
47	RG3	30.0	Roll rate gyro, channel 3
48	RG4	30.0	Roll rate gyro, channel 4
49	RS1	90.0	Aileron secondary actuator, channel 1
50	RS2	90.0	Aileron secondary actuator, channel 2
51	RS3	90.0	Aileron secondary actuator, channel 3
52	SB1	10.0	Sensor bus, channel 1
53	SB2	10.0	Sensor bus, channel 2
54	SB3	10.0	Sensor bus, channel 3
55	SB4	10.0	Sensor bus, channel 4
56	YG1	30.0	Yaw rate gyro, channel 1
57	YG2	30.0	Yaw rate gyro, channel 2
58	YG3	30.0	Yaw rate gyro, channel 3
59	YG4	30.0	Yaw rate gyro, channel 4
60	YS1	90.0	Rudder secondary actuator, channel 1
61	YS2	90.0	Rudder secondary actuator, channel 2
62	YS3	90.0	Rudder secondary actuator, channel 3

1. This effectively reduces the reliability of system 1 since the backup was removed.

Configuration (2) is recommended. Additional details are given in section 7.

4.2 RELIABILITY VALIDATION

To validate the reliability requirement in the laboratory, it must be shown that the system does indeed behave as the mathematical model. This must hold for all normal functions and all classes of failures that were hypothesized. The abstract assumptions leading to the design and realization of the reliability requirement must be shown to be closely related to the physical reality, and must be implemented correctly by the system's hardware and software.

The following sections discuss the validation of component reliability, validation of fault tolerance, failure modes effects tests, and accelerated life tests. Detailed discussion of the automated iron bird tests is deferred to section 6.

4.2.1 Component Reliability Validation

Major components are identified in the preliminary design and the reliability analysis stages. The reliability of the components in terms of MTBF are established. In order for the reliability requirement of the total system to be achievable, the MTBF for each component must satisfy this requirement. The MTBF of existing equipments can best be obtained from field service experience. If that information is not available, results of the manufacturer's reliability test program for the equipment can be used.

Reliability in terms of MTBF for newly developed equipments must be validated in the laboratory. This can be roughly divided into two processes--equipment burn-in and reliability qualification testing. Burn-in is used to assure that equipment presented for qualification testing is free of workmanship defects and other infant mortality problems. It consists of testing, analyzing all failures, incorporating corrective action, and retesting. This sequence is repeated until assurance is made that the

required reliability can be demonstrated during the reliability qualification test. The purpose of the reliability qualification test is to estimate the true MTBF of the equipment. Since constant failure is assumed, the equipment subjected to reliability qualification testing must be free of design defects or infant mortality type failures. The MTBF of an equipment can be demonstrated in the laboratory using statistical test plans such as those described in MIL-STD-781C. Depending on statistical confidence levels and discrimination ratios (criteria for acceptance and rejection), test hours of 5 to 10 times the predicted MTBF are normally required.

Measured component MTBFs should be fed back to the reliability analysis to reevaluate the total system reliability. If the reevaluated system reliability falls substantially below its required value, efforts must be made to improve the component reliability or to reconfigure the system.

4.2.2 Fault Tolerance Validation

Ultra-reliability of flight-critical systems is achieved through redundant design. The system must be able to tolerate multiple faults while maintaining undegraded flight operation. Validating the fault-tolerance and reconfiguration features of the system is the most critical step toward validating the reliability requirement of the total system.

To validate the fault-tolerance requirement, actual hardware faults must be inserted into the system. The capability of the software to detect and isolate the faults and to effect system reconfiguration must be demonstrated. Transient faults must also be included. The timing of the fault relative to the control cycle of the system must be evaluated. These processes can best be accomplished in the iron bird, where a high degree of fidelity to the flight environment is obtained through including actual system components, with software executing in a real-time, closed-loop environment.

The fundamental problem of fault-tolerance validation for a flight-critical digital system is the vast number of possible test areas when all combinations of flight conditions and multiple hardware faults must be considered.

In digital systems, software errors can only be uncovered by executing the software for a set of well-designed test cases. The following steps provide a solution to the problem.

(1) Automate testing in the iron bird.

(2) Generate a large but manageable number of test cases, from both the theoretical and practical perspectives, so that validation of digital flight-critical systems can be carried out conclusively and efficiently.

A more detailed discussion on the approach and concepts of these two areas is given in sections 6.1 and 6.2.

4.2.3 Accelerated Life Test

A typical reliability "bathtub" curve is shown in figure 20, which compares life and failure rate. The horizontal line is the failure rate predicted for mature systems following debug and early life failures. At the end of the service, life wear-out and durability failure modes appear and increase the failure rate.

Compressing the time factor by increasing cycling rates for high stress levels is the traditional method of demonstrating fatigue life. Similarly, increasing temperature and temperature cycling are the methods used to accelerate electronic systems burn-in and reduce the time required to achieve mature equipment failure rates.

Figure 21 shows the impact of accelerated testing combining cyclic testing and environmental testing. In the random portion of the curve the failure rate will be higher than normal, but this rate can be used for analytically combining failure rates in the redundant configurations of the final design.

By combining the concepts of automated cycling and increasing environmental stressing, accelerated life tests can be applied to the integrated FBW system in the iron bird. Test cases would be designed that consist of combinations of variables such as aircraft flight condition, environmental disturbance level, maneuver profile, and system model engagement. Environmental stressors such as extreme temperature and temperature cycling are applied to various components of the system. Automation would be used to step through the test cases with system and component performance monitored. Any inconsistency of system performance,

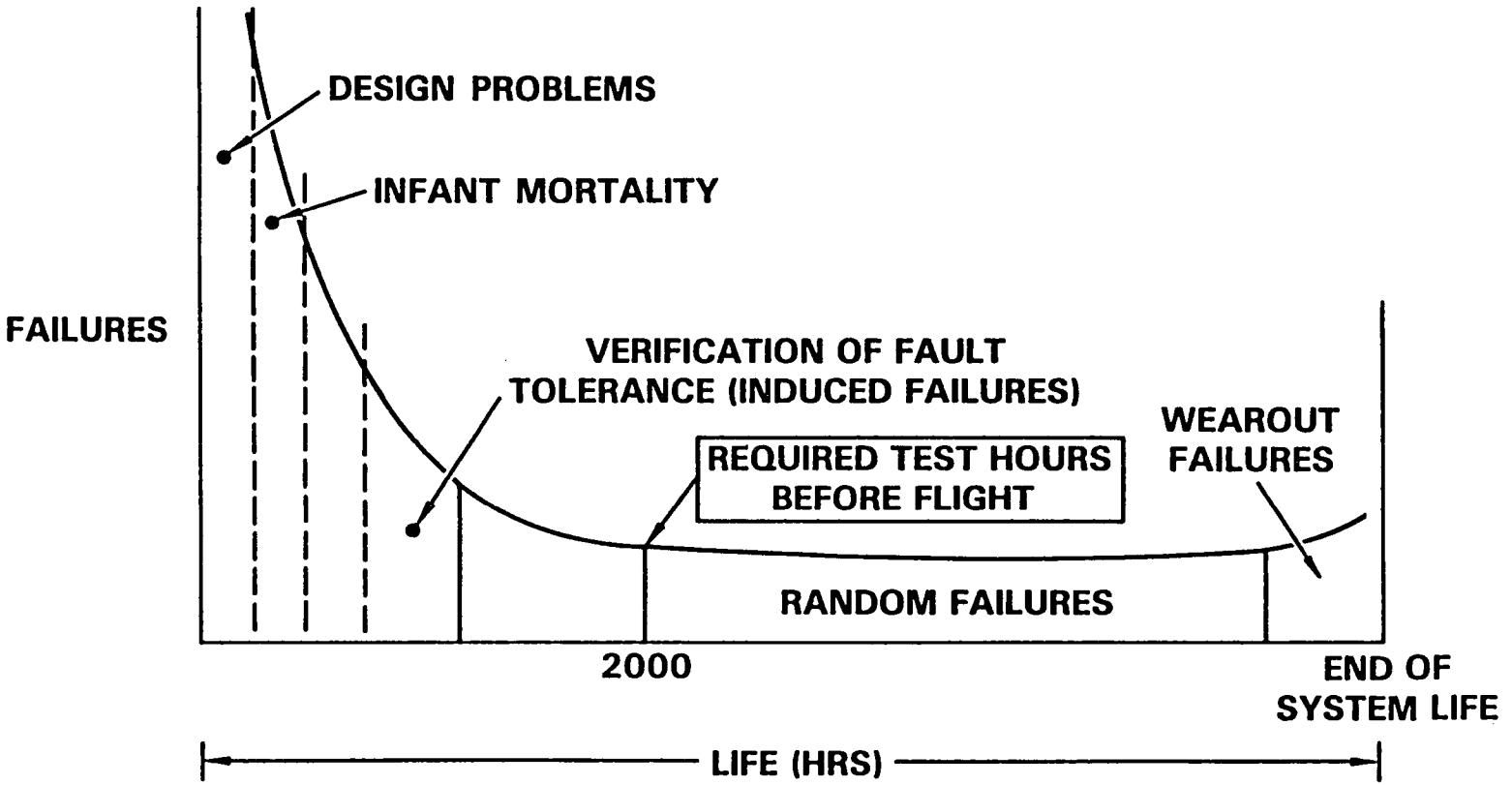


Figure 20. - Reliability life curve.

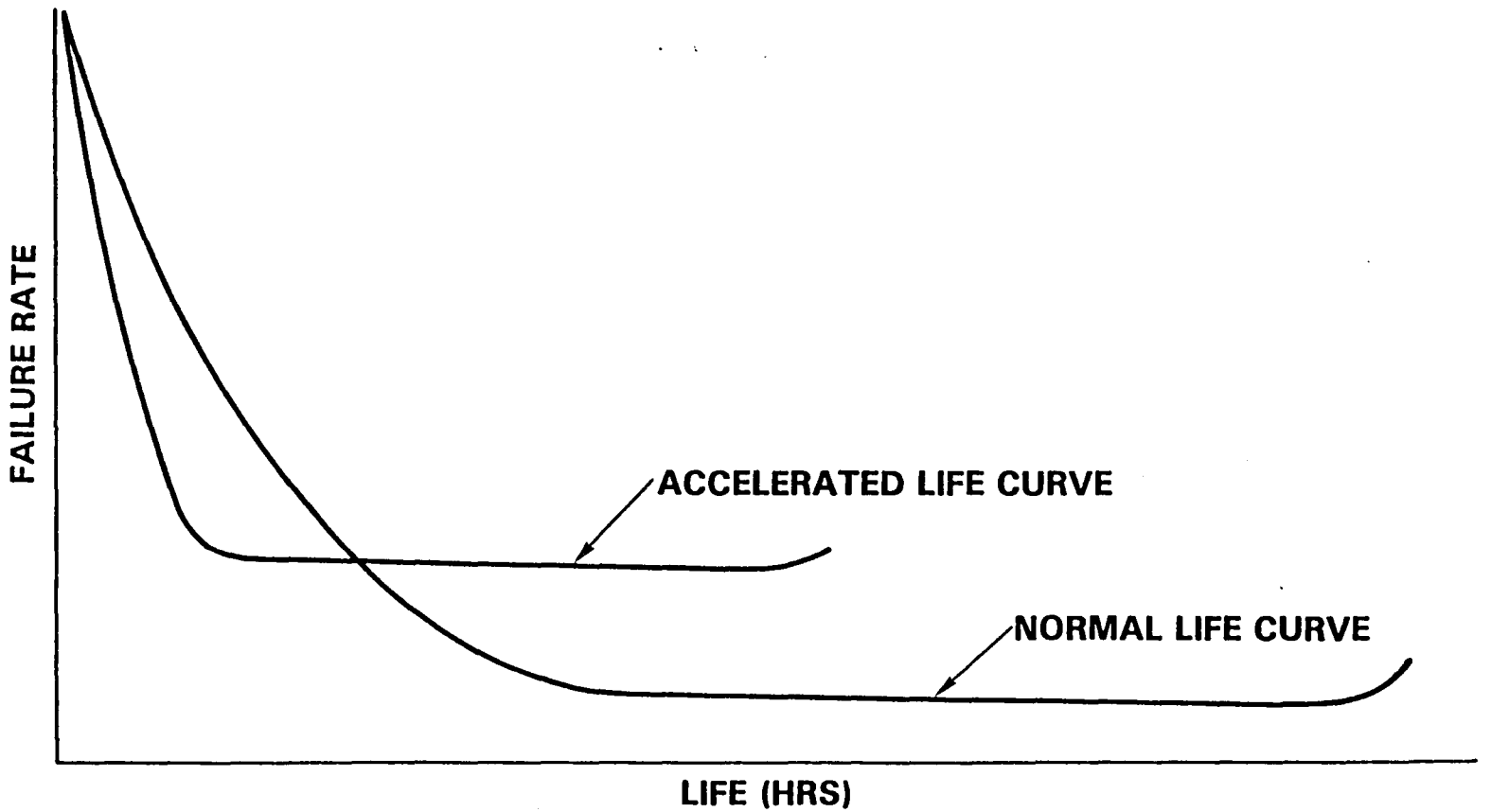


Figure 21. - Accelerated and normal life curves.

software logic error, and unexpected component failure is logged for later evaluation, and the system proceeds to the next test case. The same testing procedure can be repeated after incorporating any fixes and design changes until all design and software errors are resolved.

Using a high degree of automation and applying severe stressors for accelerated life testing would substantially reduce total test time and labor cost. It should be emphasized that this procedure identies weak points in the design of the system and eliminates infant mortality type failures rather than demonstrating overall reliability of the system. The combined efforts of analysis, component reliability tests, fault-tolerant validation, software verification, and pilot-in-loop failure modes and effects tests are necessary to validate the reliability of the ADFBW system. Accelerated life testing, however, is an important step to obtaining high confidence that the system is qualified for flight.

SECTION 5--SOFTWARE DESIGN AND VALIDATION

This section continues the description of the methodology outlined in section 3.1. The design and validation of the software is described in this section.

This section begins with a review of flight control functions. The next section reviews tools and techniques. Selected tools and techniques appropriate for flight control software are recommended.

5.1 FLIGHT CONTROL FUNCTIONS

A multichannel system in which the channels are synchronized for cross-channel comparisons requires the following functions:

Initialize computer

Run preflight built-in-tests

Recover from power transients

Initially synchronize with other computers

Exchange state data, initialize state

Monitor sensors, select signals
Manage failures, reconfigure system
Branch to rate path
Calculate control gains
Run mode and switching logic
Test intercom
Calculate control laws
Test synchronization
Test power supplies
Run output test
Perform CPU self-tests:
    Processor self-test
    Parity checker test
    Memory sum check
    Memory addressing test
    Watchdog check
    Scratchpad sum
    Sum check of constants
Test program counter
Annunciate system status
Select output signals
Monitor actuator response
Frame synchronize the computers

This software is not very complicated compared to general software systems. Indeed, many complete flight control systems have been programmed by a few control engineers. Most of these functions require straight-line programs. There are only a few do-while loops that depend on external events for termination and none that depend on their own calculations for termination. These event-driven functions are for timing synchronization events and for the watchdog timers.

The executive structure assumed in the list is a simple rate tree that calls the functions in a fixed order. An alternative is to call tasks in an interrupt-driving structure. The first is usually chosen by control

engineers; the second is chosen when software experts, familiar with operating systems, do the designing. Each has advantages and disadvantages.

A rate structure requires that the functions must be allocated to the branches of the rate tree so that the timing is balanced. This makes changes difficult because the time for execution must always be checked. Other aspects of the rate tree are easily verified. Since it is deterministic, testing will lead to confidence in its correctness. With a nondeterministic interrupt structure, no amount of testing can cover all of the possibilities. The software overhead and complexity is much higher if interrupts or a nondeterministic scheduling is used. Because of our concern for validation we recommended a rate-tree structure.

If the channels run asynchronously and are not voted, all of the synchronization functions and many of the failure management functions are unnecessary. These are the functions that have been the most difficult to verify, to show as globally consistent, and to analyze for their response to hardware failures. Verifying parallel processes is orders of magnitude more difficult than verifying sequential processes.

Most of the functions are straight calculations requiring no past results. The self-tests are of this type. The mode logic and control calculations require a fixed, finite set of results from the previous invocation of the function. These may be described as finite-state machines; however, control laws are better described by block diagrams or equations.

Provisions for equalizing control law integrations are required when the channels run independently. These impose some requirements for data exchange between the channels. In general, however, all of the functions may be precisely described and verified. Flight control functions are largely independent, so that verification and testing of the individual functions carries over after the functions are integrated into the system. We will choose modes of description and verification specific to each type of function in our methodology. Our recommendations for describing the various functions are summarized in table 7 for the ADFBW architecture. The entries labeled "sequential code" will be described in a program design language.

The next section surveys tools and techniques. We then return to complete the discussion of our methodology by describing the software design, coding, and test phases.

TABLE 7. - FUNCTIONS FOR SELF-CHECKING ARCHITECTURE

Function	Description
Initialize computer	Sequential code
Run preflight built-in-tests	Sequential code
Recover from power transients	Hardware diagrams
Exchange state data for equalization or synchronization	Sequential code
Monitor sensors, select signals	Finite-state machines
Branch to rate path	Sequential code
Calculate control gains	Sequential code
Run mode and switching logic	Finite-state machines
Calculate control laws	Block diagram, equations
Perform in-flight self-tests	Sequential code
Annunciate system status	Sequential code
Select output signals	Hardware description for the remote terminals
Monitor actuator responses	Hardware description for the remote terminals

5.2 TOOLS AND TECHNIQUES FOR VERIFICATION AND VALIDATION

A tool is a computer program which performs some task that would otherwise have to be done manually. Tools may be classified as static or dynamic. Static tools examine some aspect of the specificatons, designs, or code without executing the code of the software being inspected. An example of a static tool is the set/use checker, which checks that a variable is given a

value before it is used, or if a variable has been defined and given a value, checks that it is subsequently used. A dynamic tool performs some function to aid in testing the program when the program is actually executed. A timing analyzer that monitors and records the execution time of functions and subroutines is a dynamic tool. Two columns of static tools and one of dynamic tools are listed in table 8. Descriptions of these may be found in reference 10. The first column of static tools lists those which examine a specific property; the second column are those which examine general or more extensive properties. The set/use checker is listed as a specific tool. A symbolic

TABLE 8. - TOOLS

Specific Static Tools	General Static Tools	Dynamic Tools
Circular reference checker	Accuracy analyzer	Simulations
Code comparator	Assembly code verifier	o Computer
Consistency checker	Assertion checker	o Hybrid o Testbed (iron bird)
Cross-reference checker	Documentation and construction systems	o Monte Carlo
Data base analyzer	Formal languages with syntaax analyzers	Test data generator
Flow charter		Test driver
Interface checker		Test execution monitor
Program flow analyzer	o Requirements o Specifications o Program design	Test record generator
Set/use checker	o Program code	Timing analyzer
Standards checker	Sneak-path analyzer	
Units consistency checker	Symbolic evaluator	
Unreachable code detector	Theorem prover	
	Verification condition generator	

evaluator, a tool which automatically reconstructs the boolean or algebraic equations relating the outputs to the inputs, is listed as a general tool. Many of the static tools examine global properties--those related to the program as a whole. For example, the set/use checker may search through much of the program before it can make a determination on a particular variable.

Techniques are the standards and procedures used in developing and maintaining the software package. In table 9, we distinguish between those for development and those for analysis or review. The entries are also discussed in reference 10. The list of development techniques cannot be omitted on the grounds that it is not verified and validated. Practice has shown that substantial gains in software reliability can be obtained by attention to description, documentation, and systematic development.

Some approaches to software development use an integrated set of tools including static and dynamic code analyzers, test and simulation facilities, and documentation aids. Other methodologies use a formal design language based on the constructive approach outlined by Dijkstra, and have elaborate facilities for recording design progress and documentation on a large computer. An integrated approach is also taken for verification systems. In addition to static analysis and testing, verification conditions and symbolic executions are used. These systems show great promise, but it is currently difficult to determine the extent of the error coverage and the completeness of the entire verification procedure.

The best engineering practice is to work at the lowest level of technical sophistication that will solve the problem. Flight control systems need not be obscure; the software functions to be performed are elementary. A verification and validation methodology with tools and techniques selected from the foregoing lists need not be elaborate. It must, however, be precise and complete. The next section outlines a methodology specific to a flight control system.

5.3 SOFTWARE DEVELOPMENT

Our methodology follows traditional lines, but with an emphasis on the simplicity and clarity of structure. Everything must be made "public" so

TABLE 9. - TECHNIQUES

Development	Analysis
Abstractions and hierarchies to reduce complexity	System concept review
Constructive design approaches	Software design review
Data flow graph, structure chart	Critical design review
Descriptions	Qualification audit
o Charts --HIPO --Flow charts o Languages --Requirements --Specification --Program design --Program code o Petri nets o LOGOS	Checkout testing Singularities and extremes testing Integration testing Validation testing Symbolic execution Inductive assertions
Design guidelines, test guidelines, coding guidelines Design standards, coding standards Functional capabilities list Organization as finite automata	Proofs of data structures State transition proofs Recursion functions
o Parnas modules o SRI International formal modules	Fault-tree analysis
Axiomatic specifications	

that the details may be reviewed and checked. The completeness of the configuration and its response to all anticipated events are set out for everyone to see. We cannot use the "wizard" who specifies, designs, and tests segments of code with little explanation and documentation. Flight control design is often done by three to four very experienced control engineers who collaborate and cross-check as the work proceeds. Proven software modules and functions are reused for efficiency and accuracy.

5.3.1 Software Design

The design of the software is based on the program performance specification and the interface design specification prepared during the requirements phase (refer to section 3). The output of this phase is the program design specification (PDS) and the data base design. The PDS tells how the program works by describing functionality and interfaces. It contains memory and time allocations and programming guidelines. The data base design describes all shared data. The specifications can follow MIL-STD-483 but require precise specification of functions.

The software specifications, if complete, may be quickly transformed into the design for the software code. We have found that a hierarchy-input-plus-output (HIPO) format is satisfactory for flight control functions, and we recommend it. The data flow between modules is clearly identified, state variables are precisely defined, and the program, expressed in any convenient program design language, is easy to review, to subsequently code, and to verify that it meets the software specifications.

We have used an informal Pascal-like pseudocode as a program design language. It appears sufficient to represent the flight control functions unambiguously. Formal program design languages have been recommended. These have syntax and other features that may be automatically checked. Eventually, Ada will be used in this capacity. However, the fundamental simplicity of flight control functions allows these aspects to be checked with confidence by walk-through reviews.

During this phase a software functional design review is held. This formal review of the detailed software design, documented in HIPO charts, will be conducted prior to the start of coding. The object is to demonstrate by

walk-through presentations that the software design satisfies the software specifications. Analysis to show the completeness and consistency of the data flow between modules will be presented. The global consistency of the built-in-tests and the failure management functions will be verified by a lattice showing the dominances of the tests. The operations and failure modes analysis of any synchronization schemes will be reviewed at this time.

5.3.2 Coding

The coding is done directly from the HIPO design charts. The code is verified by inspection and by testing. Prior to integration, each module will be informally tested to verify that it performs according to the design and fulfills the software specifications. The testing approach will be chosen to be suitable for the function of the module.

The following test procedures are suggested for the classes of functions of the flight control system:

(1) The executive structure for flight control is not complicated. It is feasible to test every path and every branching action.

(2) Control mode switching modules may be designed as finite-state machines. These tests should verify that the processing of input data to yield transitional events and outputs is correct for each state and that all state transitions occur as specified.

(3) A control law calculation may be verified by showing that a reasonable segment of the required frequency response is achieved. Extreme values of inputs and even stressed values of inputs must be tested to show that limiters and overflow provisions are operating correctly. Gain schedule calculations may be thoroughly tested.

5.3.3 Testing the Code

Each module can be exhaustively tested. All transitions of the finite-state machines can be checked. The built-in-tests and many other functions are copied from previous systems and have already undergone complete testing. Most of the functions are independent of everything else in the system, so their correctness will hold after integration if the input/output data flow is correct and the state variables and constants are correctly

maintained. A peer review of the code should be made to look for unusual or complicated constructions.

(1) Most of the built-in-test functions are elementary and may be thoroughly tested for normal operation and simulated failures. Complete testing of wrap-arounds and other exogenous procedures must wait for system integration.

(2) The algorithms for redundant signal selections may be represented as finite-state machines and as such may be tested to assure complete confidence in their correctness.

(3) There are also only a finite set of possibilities for the states of the failure detection and reconfiguration control structure. These modules may be tested to assure complete confidence.

Tests are run after software modules are integrated to show that the state data is correctly preserved and that the data flow between modules is correct. Simulated failures to verify the response of the built-in-tests, the failure management structure, and the status annunciation are used.

Tests are conducted after system integration to complete the verification of the flight control functions. This is discussed in section 6.

SECTION 6--SYSTEM PERFORMANCE AND FUNCTIONAL VERIFICATION TESTING

6.1 OVERVIEW

This section describes our methodology for validating the functions and reliability of a system. Section 4 addressed the analysis and prediction of reliability. It recommended using a fault-tree description of the various elements. Validation by testing can be divided into two areas, as shown in figure 22. In the area of components and subsystems, it is feasible to run life tests to statistically validate MTBF predictions. In the system test area, it is not possible to run life tests. Instead, integrated system (iron bird) tests are used to verify the fault tolerance achieved by redundancy. These results, when combined with the component MTBFs, permit extrapolation of the complete system's reliability.

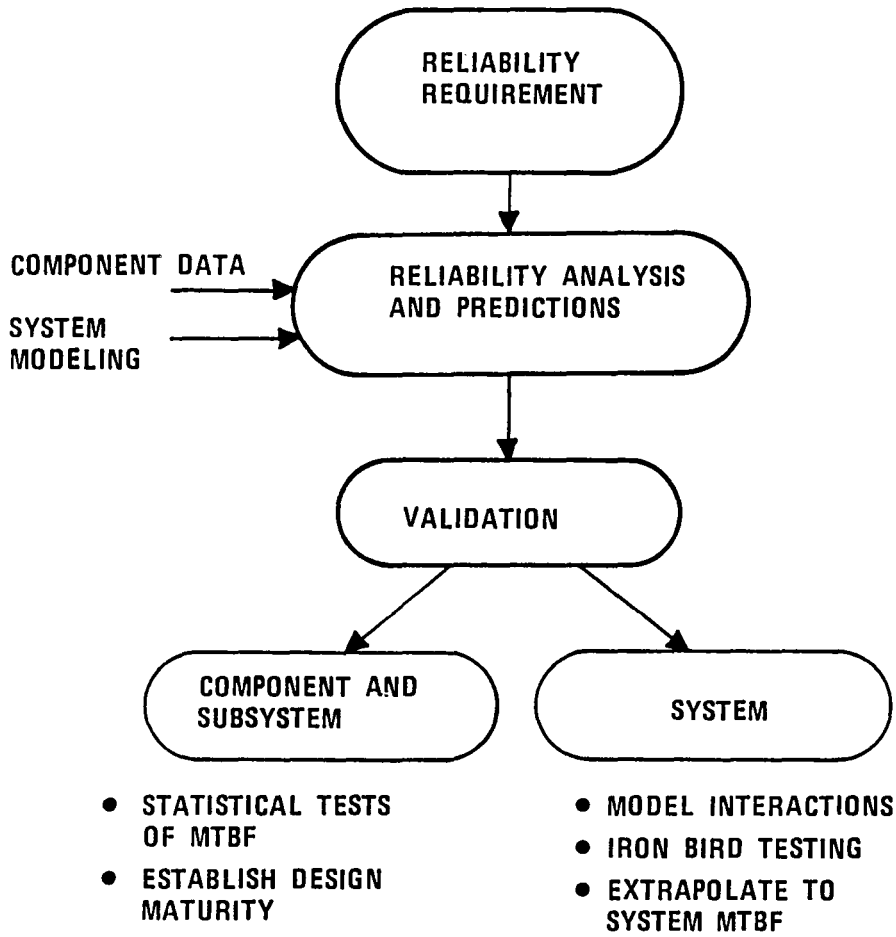


Figure 22. - The validation of an ultra-reliable system depends on indirect testing.

Section 6.2 discusses the design of test cases for integrated system testing. The finite-state machines used to specify the functions are now used in the validation testing phase. By virtue of using a finite-state machine, various "automata theoretic" results are available for specifying valid and reliable tests. Next the fault tree model is used to identify the various combinations of faults that need to be tested to demonstrate the predicted fault tolerance. Section 6.3 addresses the issue of automating the testing on the iron bird. This section concludes with a discussion of flight test plans.

6.2 DESIGN OF TEST CASES

Ultra-reliability of flight-critical systems is achieved through redundant design. The system must be able to tolerate multiple faults while maintaining undegraded flight operation. Validation of the fault-tolerance and reconfiguration features is the most critical step to the validating the reliability of the total system. These processes can best be accomplished in an iron bird, in which a high degree of fidelity to the flight environment is obtained by including actual flight hardware operating in a real-time, closed-loop simulation. The fundamental problem of fault tolerance validation is the vast number of test cases when all possible combinations of flight conditions and multiple hardware faults are considered. Effective testing requires:

(1) A methodology using both theoretical and practical perspectives to define a manageable set of test cases.

(2) Automating the testing as much as practical.

Consider the set of conceptual states shown in figure 23. It can be used to visualize possible test cases. The universe of test possibilities is first divided into two areas: everything operable and some element failed. The operative region can be described by one or more finite-state machines. The effect of various failures on the system can be described with a fault tree. The "failed element" region includes failures for which a reconfiguration strategy was designed (i.e., switches out failed element), as well as failures external to the system which are to be tolerated (i.e., loss of a hydraulic power supply). The management of redundant elements can be described using finite-state machines.

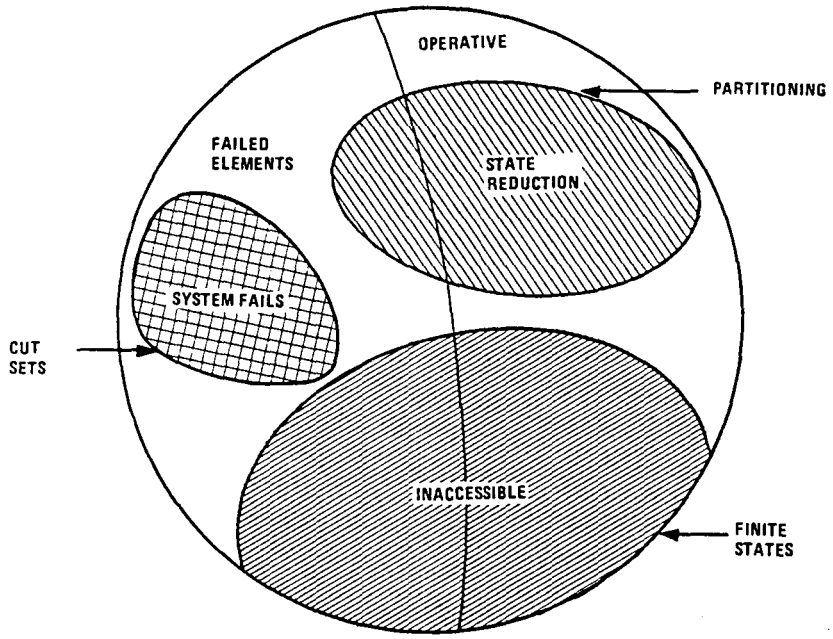


Figure 23. - Conceptual system states.

Three cross-hatched areas represent regions where analysis can be used to reduce the number of test cases:

(1) System failed. - These cases, identified as cut sets in the fault tree, are discussed in more detail below.

(2) Inaccessible. - These states represent combinations of modes, flight conditions, or environments that are mutually exclusive and do not need to be tested.

(3) Partitioned. - this area represents states that can be partitioned such that all combinations do not need to be tested. This could involve use of hierarchies of finite-state machines or partitioning on the basis of control modes, etc. The next section outlines a method for designing test cases based on the finite-state models.

6.2.1 Use of Finite-State Machines

As discussed in section 2, many of the "control structures" specifying mode switching, signal selection, and failure management can be described as finite-state machines. Using finite-state machines in system and software design provides a unique link between the specification, the design, and the testing phases. Automata theoretic results can be used to show the correctness of a control structure. The method consists of the following steps (ref. 22):

(1) Estimate the maximum number of states in the correct design.

(2) Generate test sequences based on the design.

(3) Verify the responses to the test sequence generated in step 2.

In step 1, the estimate can be based on the design. In step 2, test sequences are generated that exercise all the state machines (obviously these sequences are not unique). In step 3, the input sequences and their responses can be represented as "path programs." The correctness of these programs can be established by a walk-through procedure based on the specification. Since we have assumed the specification is not "executable," it is not possible to totally automate this step.

The above method is both valid and reliable for checking the control structure. The detectable error classes include:

(1) Missing states

(2) Extra states

(3) Transfer errors

(4) Operation errors

It is particularly useful that theoretical results are available to extend the method to designs based on multiple finite-state machines and hierarchies of finite-state machines. For flight control a hierarchy of structured finite-state machines keeps the number of states of each machine at a tractable number. Some aids in deriving a hierarchy of structured finite-state machines from a given finite-state machine are given in reference 23. using hierarchies of finite-state machines is a tremendous asset in reducing the number of combinations.

6.2.2 Use of Fault-Tree Analysis

The fundamental problem in the fault-tolerance test is the vast number of conceivable test states as multiple hardware faults are considered. To test all possible combinations of component failures for a large, complex system is clearly impractical. As a result, a more economical but theoretically sound and conclusive approach is needed to design the test states for multiple faults. We will describe such an approach based on the fault-tree analysis used to calculate the theoretical reliability.

This paragraph summarizes the fault-tree method. A more elaborate discussion is found in section 4.1.1. Basically, the fault tree is a top-down method of describing the failure of a system. The top event is the occurrence of total system failure, modeled by logical combinations of the failure of its associated subsystems. This process is repeated for structuring the subsystems until reaching the lowest event--the failure of the basic components. Boolean expressions are generated that list all possible minimal combinations of component faults leading to total system failure. Each of these fault combinations is known as a minimal cut set. The probability of total system failure is computed by combining the probability of occurrence for each minimal cut set.

The fault-tree analysis of a system can be used to develop test states for validating fault tolerance. The purpose of fault-tolerance testing is not to prove that the system fails when the fault tree predicts it will, but rather the converse. The purpose of this testing is to establish that the system works correctly when the fault tree predicts it will. To establish the former, the various fault combinations that make up minimal cut sets are used as test states, and the failure of the system is expected. This testing would demonstrate that the system fails at least as often as the fault tree predicts. The important case to establish is the latter. In this case, various combinations of faults that do not contain cut sets are used as test states. The system is expected to work correctly for all of these combinations. If it does, then this testing demonstrates that it works at least as often as the fault tree predicts.

We define a test set as a set of component failures that contains no cut set. This means that the fault-tree analysis predicts the system should not

fail in the face of failures contained in any test set. A maximal test set is defined as a test set that is not contained in any other test set. This means if any component failure is added to a maximal test set, the resulting set is a cut set. The relationships among these sets of component failures are shown in figure 24.

One way of generating test sets is to consider the maximum components in each cut set that the system can tolerate. This is simply one component less than the total components in a cut set. The number of such test states is equal to the total number of components in the cut set. This approach, however, does not consider the combinations of the elements in one cut set with the elements of the other cut sets. The effects of these fault combinations are unknown if not tested. The maximal test sets do consider cross-combinations among several cut sets and so, in general, contain more elements than just "all but one component" from a cut set.

These definitions allow us to state that a system works at least as often as its fault tree predicts if it works correctly for each combination of component failures in a test set. However, to test all the combinations in the test sets is still impractical for large, complex systms. We would like to use the maximal test sets for testing purposes in a way analogous to using minimal cut sets for reliability analysis. This requires the following assumption:

<u>If a system fails under a given set of component faults, then it will fail under the given set of component faults plus any additional component faults</u>.

This assumption is necessary to avoid having to test all combinations of faults in the test sets. With this assumption, it suffices to demonstrate the capability of the system to operate under the combination of faults in each maximal test set.

These ideas will be made more concrete by means of an example. The example considers the 25 most likely minimal cut sets found in the reliability analysis of the ADFBW system, together with the S-3A systems (see table 10). Maximal test sets--the largest sets not containing a cut set--will be found for this example. Cut set numbers 1, 2, 6, 11, 12, and 13 are independent minimal cut sets in that each of these sets has no elements in common with any other minimal cut set. Cut set number 11 is

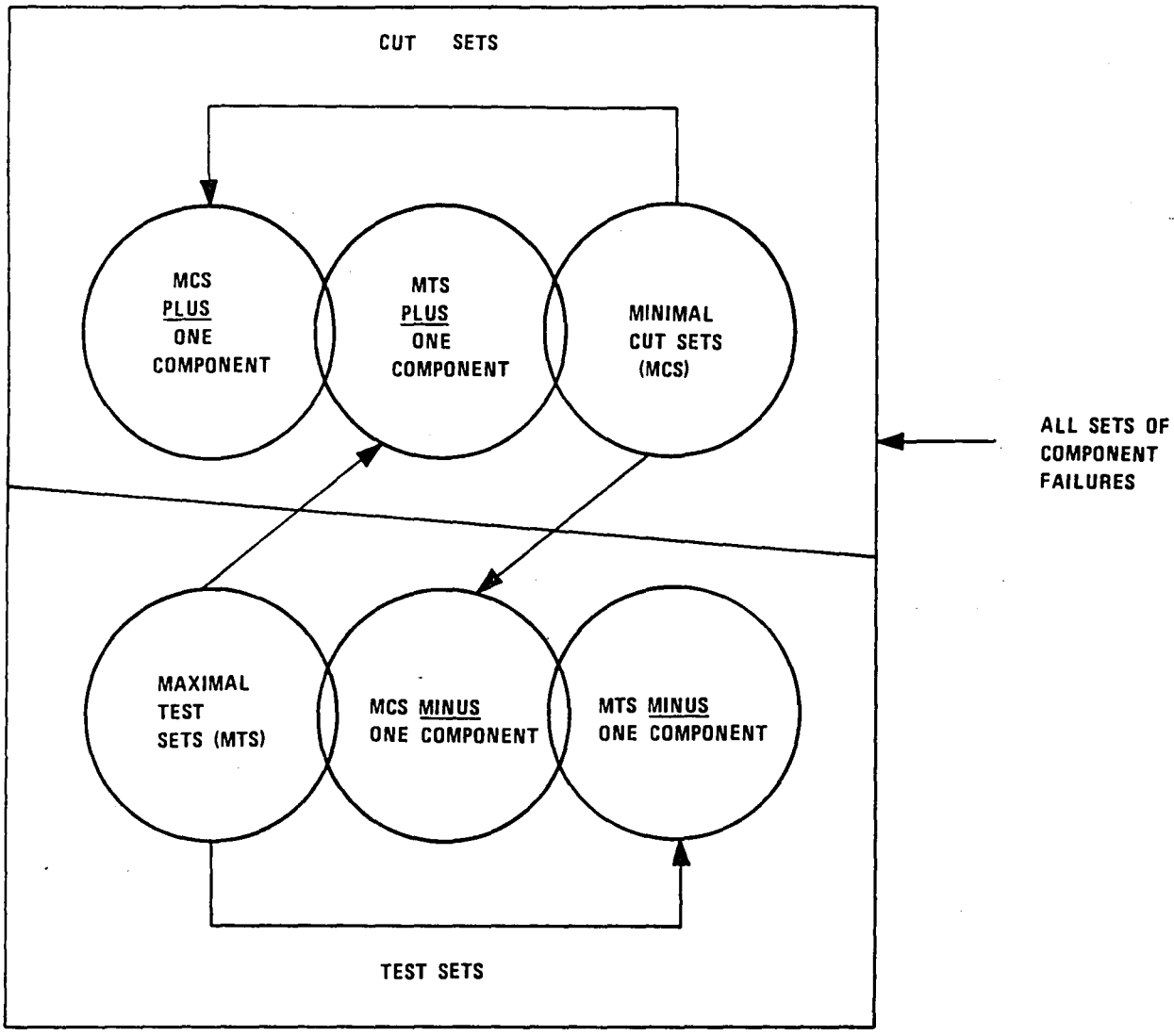


Figure 24. - Any combination of component failures is either a cut set or a test set.

typical. Three elements are contained in this cut set--the three redundant channels of the aileron secondary actuator (RS1, RS2, RS3). If all three servos fail, the roll channel fails, resulting in loss of aircraft control. The maximum number of aileron secondary actuator channel faults that the system can tolerate is therefore two. Since aileron secondary actuator channels do not appear in any other minimal cut sets, any maximal test set must contain exactly two aileron secondary actuator channel faults. Similarly, each maximal test set must contain exactly one element from cut

TABLE 10. - MINIMAL CUT SET DATA IN DESCENDING ORDER OF PROBABILITY

Cut Set Number	Maximum Failure Probability	Components Contained in Set	Description of Mnemonic
1	0.99989848E-08	ENG1 ENG2	Engines
2	0.15619062E-10	CC1 CC2 CC3	Computer channels
3	0.99984777E-12	GEN IDG1 IDG2	APU generator, integrated drive generators
4	0.99984777E-12	AAPU IDG1 IDG2	APU accumulator, integrated drive generators
5	0.99984777E-12	APU IDG1 IDG2	Auxiliary power unit, integrated drive generators
6	0.99984777E-12	BPMP PMP2 PMP1	Backup pump, pumps
7	0.75346635E-12	AC1 AC2 AC3	Air data computers
8	0.75346635E-12	AC4 AC2 AC3	
9	0.75346635E-12	AC4 AC1 AC2	
10	0.75346635E-12	AC4 AC1 AC3	
11	0.72889996E-12	RS1 RS2 RS3	Aileron secondary actuators
12	0.72889996E-12	PS1 PS2 PS3	Elevator secondary actuators
13	0.72889996E-12	YS1 YS2 YS3	Rudder secondary actuators
14	0.82801821E-13	AC1 AC2 SB3	Air data computers, sensor bus
15	0.82801821E-13	AC4 AC2 SB3	
16	0.82801821E-13	AC4 AC1 SB3	
17	0.82801821E-13	AC4 AC1 SB2	
18	0.82801821E-13	SB4 AC1 AC2	
19	0.82801821E-13	AC1 SB2 AC3	
20	0.82801821E-13	AC4 SB2 AC3	
21	0.82801821E-13	AC4 SB1 AC3	
22	0.82801821E-13	AC4 SB1 AC2	
23	0.82801821E-13	SB1 AC2 AC3	
24	0.82801821E-13	SB4 AC2 AC3	
25	0.82801821E-13	SB4 AC1 AC3	

set number 1 (since it has two elements) and two elements from cut set numbers 2, 6, 12, and 13. These cases are listed in table 11.

The remaining minimal cut sets may be divided into two groups. One group is cut set numbers 4, 5, 6 and the other group is numbers 7-10, 14-25. The two groups have no elements in common. For both groups, their contribution to maximal test sets is determined readily by inspection. The first group is illustrated in the Venn diagram of figure 25. Since each of these cut sets has the elements IDG1 and IDG2 in common, the maximal test sets fall into one of three cases:

(1) All the elements except IDG1 are included

(2) All the elements except IDG2 are included

(3) Only IDG1 and IDG2 are included

TABLE 11. - MAXIMAL TEST SETS FOR THE EXAMPLE ARE CONSTRUCTED AS THE UNION OF ONE SUBSET FROM EACH OF THE EIGHT INDEPENDENT GROUPS

Cut Set Number	Intersection of the Cut Sets with Maximal Test Sets (failed components)
1	{ENG1}, {ENG2}
2	{CC1,CC2}, {CC1,CC3}, {CC2,CC3}
6	{BPMP,PMP1}, {BPMP,PMP2}, {PMP1,PMP2}
11	{RS1,RS2}, {RS1,RS3} {RS2,RS3}
12	{PS1,PS2}, {PS1,PS3}, {PS2,PS3}
13	{YS1,YS2}, {YS1,YS3}, {YS2,YS3}
4,5,6	{GEN,AAPU,APU,IDG1}, {GEN,AAPU,APU,IDG2}, {IDG1,IDG2}
7-10,14-25	{AC1,AC2,SB1,SB2}, {AC1,AC3,SB1,SB3}
	{AC1,AC4,SB1,SB4}, {AC2,AC3,SB2,SB3}
	{AC2,AC4,SB2,SB4}, {AC3,AC4,SB3,SB4}

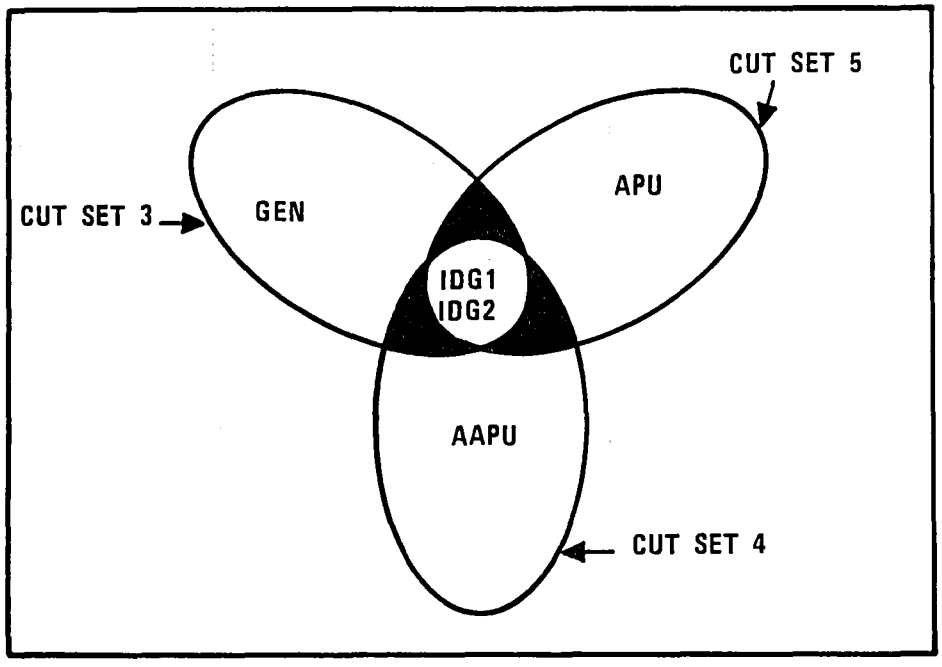


Figure 25. - There are only three groupings of the elements of cut sets 3, 4, 5 into maximal test sets.

The second group, consisting of cut sets 7-10 and 14-25, has a symmetry which can be exploited. Cut sets 7-10 are all combinations of three faults out of four air data computers. Cut sets 14-25 are all combinations of one sensor bus failure and two faults out of the other three air data computers. (As an aside, two sensor bus failures and one air data computer fault is also a minimal cut set, but is less likely than the 25 shown in table 10.) The maximal test sets must contain failures of two air data computers and the two corresponding sensor buses. There are six such combinations. All the maximal test sets may be found in table 11.

The total number of maximal test sets equals the product of the number of elements in each independent minimal cut set and the number of combinations from each dependent group of minimal cut sets. For the example problem, this is

$$2 \times (3)^5 \times 3 \times 6 = 8748$$

In contrast, the total number of combinations of failures for the 30 components in table 10 is

$$2^{30} \approx 10^9$$

Testing only the maximal test sets results in tremendous savings. Moreover, it is sufficient to guarantee the performance of the system under the assumptions stated above.

It appears feasible to construct a computer program to generate all the maximal test sets from the minimal cut sets. Development of such a program, however, was beyond the scope of this program. Additional research and development efforts are needed to establish this feasibility.

Considering only maximal test sets offers large savings in the number of required test states. Additional savings are realized if the system can be partitioned into groups on the basis of failure modes and effects analysis, or some other such analysis. The danger here lies in omitting some failure mode from the fault-tree analysis and then partitioning the system on the same basis. Still further reductions in the amount of testing may be obtained by calculating the probability of occurrence for each maximal test set. If this probability is sufficiently low, then the set can be eliminated as a test state. This elimination amounts to saying that the failure combination is so remote that the system is allowed to fail in response to that combination. This results in a slight decrease in the validated system reliability, in return for a reduction in the required testing. Together, these methods will produce a set of well-defined test states for validating the fault-tolerance of the ADFBW system.

6.3 AUTOMATED IRON BIRD TESTING

The vehicle system function mockup, commonly known as the iron bird, is a key element in the system verification, validation, and flight qualification. It exposes the real system to the most realistic environment by including actual hardware and interfacing subsystems (e.g., electrical, hydraulic, flight control systems) in the testings. This allows subsystem interface problems to be resolved and the performance of the integrated system and the pilot interface functions to be evaluated. Above all, it provides a high degree of confidence in the verification and validation test results.

6.3.1 S-3A Iron Bird

The vehicle systems functional mockup, as the S-3A iron bird is formally known, was designed and used for the S-3A system development and evaluation test. The overview of the facility is shown in figure 26. Briefly, the S-3A iron bird is a full-size, spatially correct mockup of the S-3A aircraft. It uses a structural steel framework to support the aircraft functioning systems. Within this steel framework are the various aircraft subsystems. Sufficient aircraft structure is incorporated to allow deflection of brackets, attach points, etc, thereby providing the same functional environment found on the airplane. The subsystems included in the iron bird are: the flight controls, hydraulic power generation, alighting and launching gear, wing and fin fold, and bomb bay door drive. Each of the subsystems is comprised of many individual elements, for the most part in their operational configuration. During the S-3A development, the iron bird was coupled to a moving base flight simulator so that the piloted experiments could be conducted in the most realistic fashion short of actual flight.

Functional testing of system components using this iron bird is proposed to be an integral part of the total advanced flight control system development program. The S-3A iron bird provides an ideal testbed for directly comparing the existing flight control system with the system to be developed. Because the simulator is already designed and built, cost to the program will be minimized. All interface problems between the existing iron bird and the FBW components will be investigated and remedial action recommended.

Our development program will parallel the previously conducted tests on the S-3A iron bird; however, there will be increased emphasis on automatic checkout in all system states. The tests can be grouped in the following categories:

Intersystem interaction effects

Subsystem performance and evaluations

Complete integration

Endurance cycling

Simulated component failures

Automated system checkout

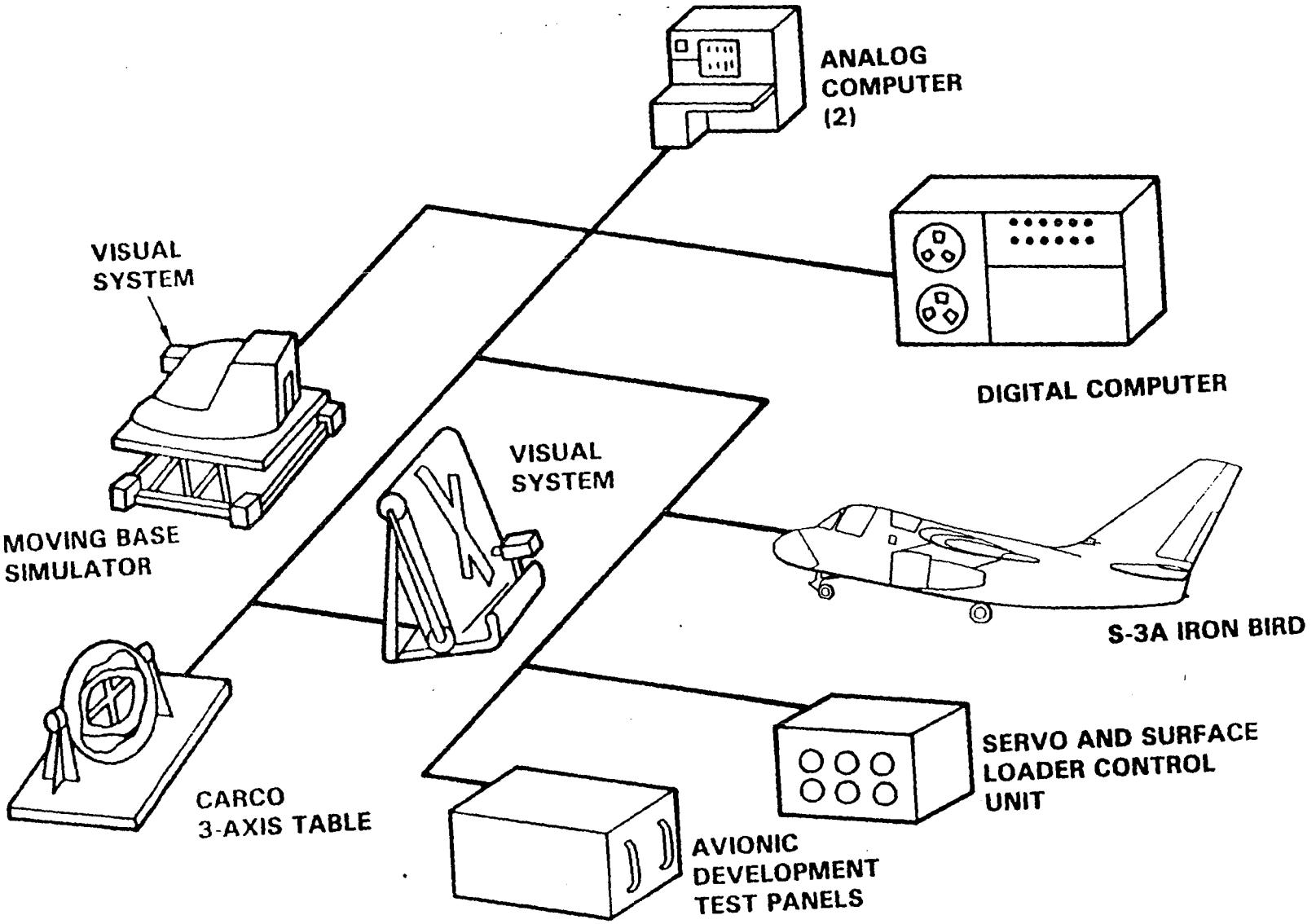


Figure 26. - S-3A iron bird overview.

In the S-3A program all reasonable system failures (subsystem and individual components) were examined during the span of the program. Types of failures demonstrated were:

Engine-out conditions during various flight modes

Valve failures--various subsystems

Broken linkages

Mis-rigging mechanical and electrical components

Failed hydraulic lines

Loss of fluid

Electrical control logic malfunctions

Jammed control linkages

The preceding list of S-3A test programs indicates the scope of work possible on the iron bird simulator. Many of these tests were so critical that if performed on a flight vehicle they could have jeopardized the airplane and pilot/crew. However, when performed on a completely integrated ground simulator such as the iron bird, this costly risk was eliminated.

6.3.2 Automated Testing

Section 6.2 discussed the development of test cases. Fault insertion using the iron bird is an important part of validating the redundancy management and verifying the reliability prediction. Because large numbers of tests need to be run, benefits are derived by automating this process. Automation provides two benefits:

(1) Time compression of test schedules

(2) Repeatability of complex test procedures

Automation can aid in performing complicated tests that require execution of precise operations in specific sequences, including the necessity for repeatedly performing tests under different test/failure conditions. Incorporating automation allows the performance of these tasks with a minimum of supervisory manpower needed to monitor rather than perform the test. A sample test plan for iron bird testing is contained in appendix B.

A robotic system concept for automatically controlling the iron bird is presented in this section. The design is highly modularized and can be

integrated into existing iron bird facilities with modest additions or modifications. The four fundamental function modules of the robotic system are the executive procedure, cockpit robotic computer, system monitor, and fault-insertion/environmental-input controller, as depicted in figure 27.

Executive procedure. - The test state matrix consists of flight conditions, multiple hardware faults, environmental profiles, and control modes engagement, stored in the executive procedure module. The executive procedure module is the central controller of the robotic system for stepping through the test state matrix. Information is sent to each function module to set up the iron bird configuration for automatic checkout of all test states.

Cockpit robotic computer. - The cockpit robotic computer emulates the pilot actions to control the airplane for those test states requiring manual interactions. Nominal command profiles such as airspeed, aircraft attitudes, trim settings, and procedures for modes control engagement for all maneuver profiles are stored in the robotic computer. These commands are converted into force commands, which are sent to specially built actuators and switches located in the cockpit, to generate command motions to the control stock, pedals, trim switches, and control mode switches for controlling the airplane in the simulator. Figure 28 shows a typical actuator interface for this concept. All information available to the crew such as fault annunciations, aircraft states, and command reference errors is fed back to the pilot model for corrective actions. Pilot dynamic models such as those reported in reference 4 are stored in the computer to simulate the pilot's response to various tracking tasks. This allows automated testings for the FBW system, in which the pilot's input is an important part of the system's design.

System monitor. - The system monitor records all test results for post-testing data processing and for in-line monitoring of system status. Outputs from all subsystems, major components, aircraft simulator, fault annunciators, and simulated pilot's commands are collected by the system monitor. Total test times and component failures are logged for analyzing component reliabilities. Internal computer computations, such as the outputs and state transitions of the finite-state machines, are also

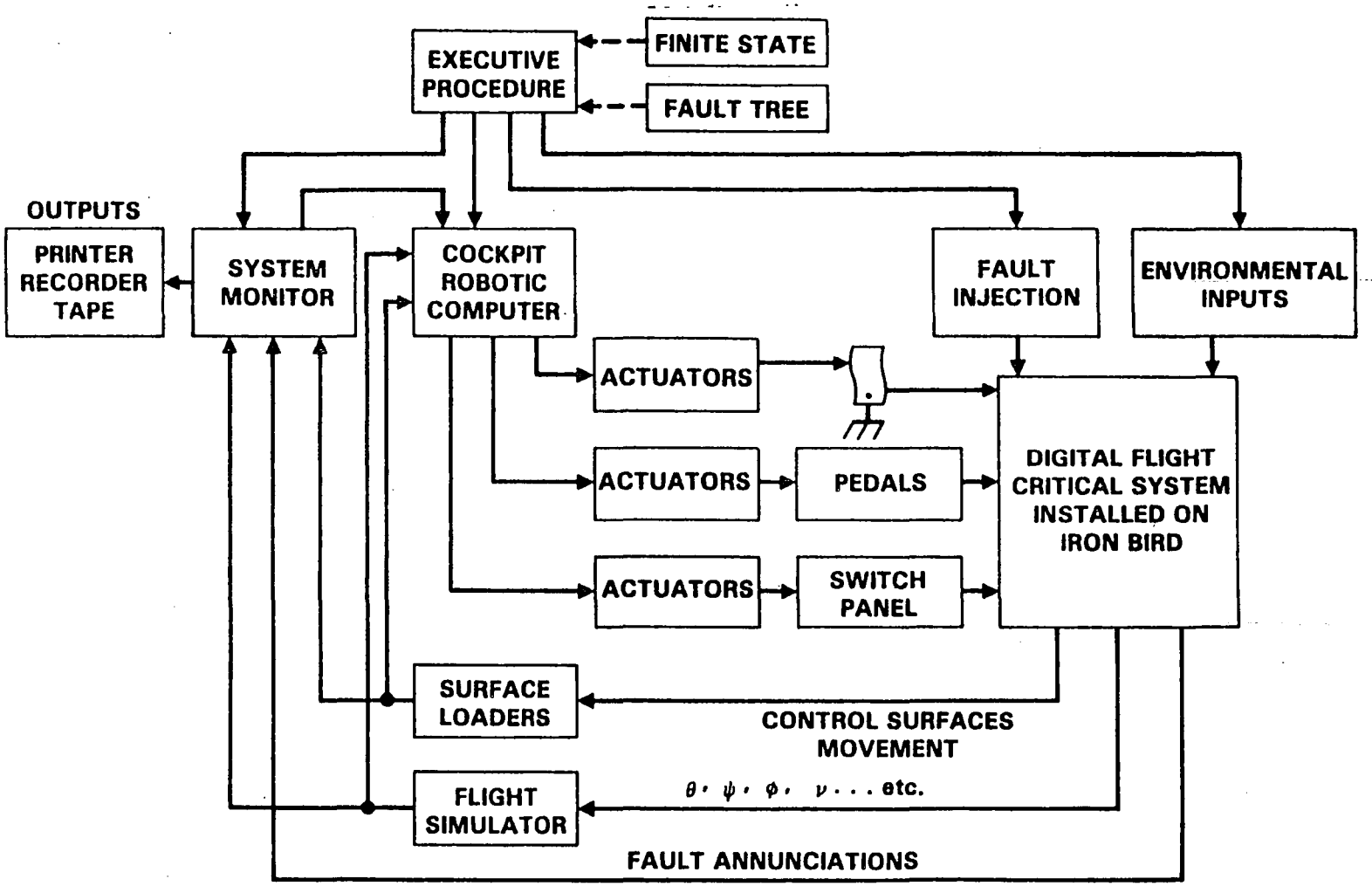


Figure 27. - Robotic system concept.

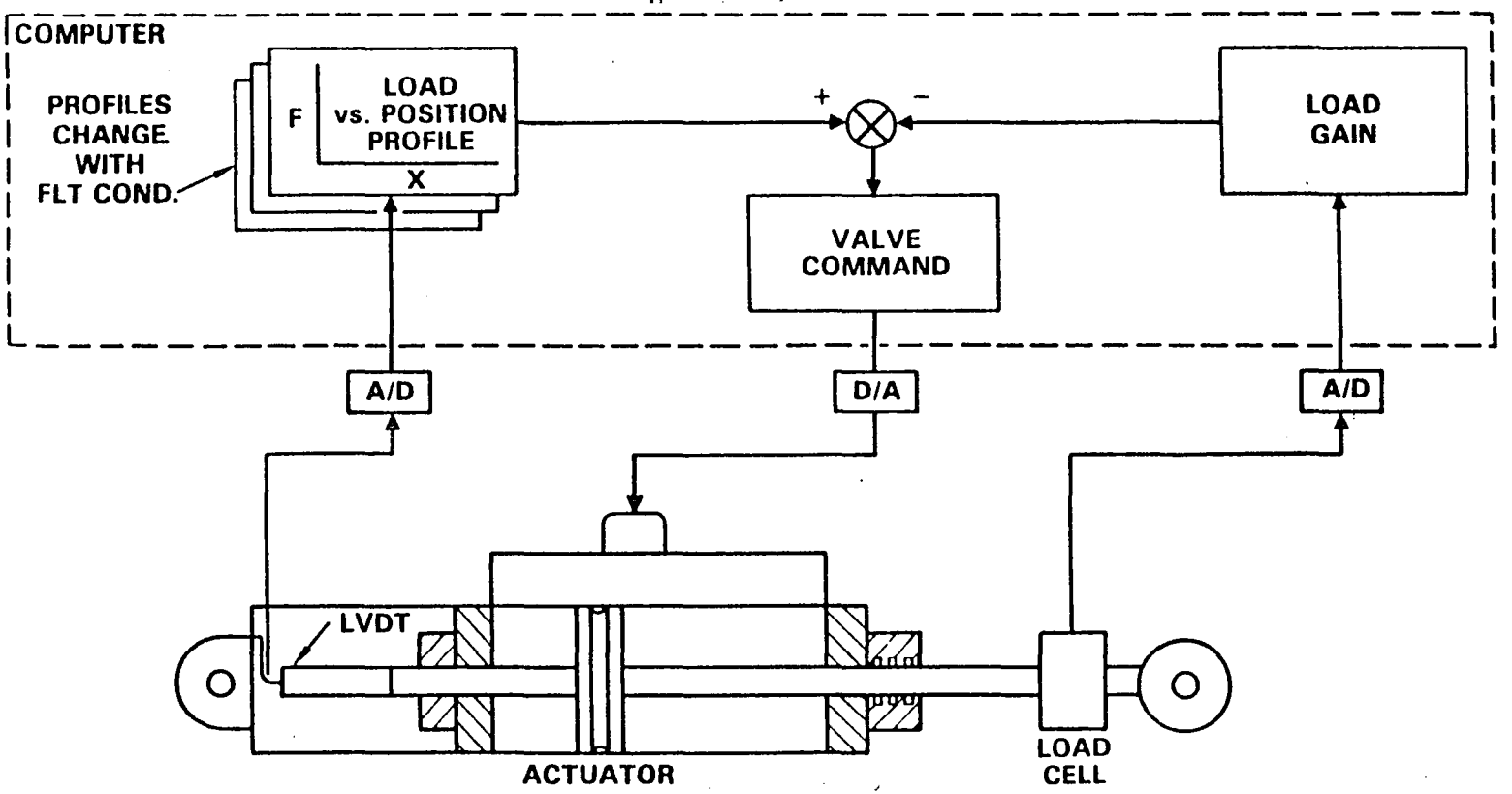


Figure 28. - Actuator interface.

recorded by the system monitor for software verification. Data reduction after each test run can be done in a separate host computer for evaluating system performance and for comparing with predicted outputs. This process detects and isolates design and implementation errors.

Fault insertion and environmental input controller. - This module provides interfaces with the fault injection hardware and the environmental control units. Multiple hardware faults can be inserted into the system by using relays for shorting of wires, solenoid switches for controlling hydraulic systems, switches for controlling electrical power systems, voltage regulators for inserting hardovers into sensors, and fault injector boards for injecting hardware faults into the FBW computers. Local insulated enclosures can be built around various subsystems for extreme temperature and temperature cyclings. Automatic setting of this hardware is provided by this function module.

Implementation of the robotic system. - Current desktop microprocessor-based computers offer the following performance:

(1) Active memory	2M bytes
(2) Expandable mass storage	Up to 125M bytes
(3) Clock frequency	8 MHz
(4) Word length	16 bits output with 32 bits internal architecture
(5) I/O buses	5 to 10 I/O cards which use popular I/O buses such as the IEEE 488, RS-232C
(6) I/O rate	1M bytes/sec
(7) Programming language	HOL

The high computation rate, large I/O interfacing capability, large available memories, and the relatively low cost of these processors are ideal for the applications of real-time automated testing of complex flight control systems. Also, the use of structured HOLs such as Pascal will reduce test software development cost.

The low cost of these microprocessors allows acquisition of several of these units for a modularized design of a robotic system, as illustrated in figure 29. The microprocessors are networked together with the executive

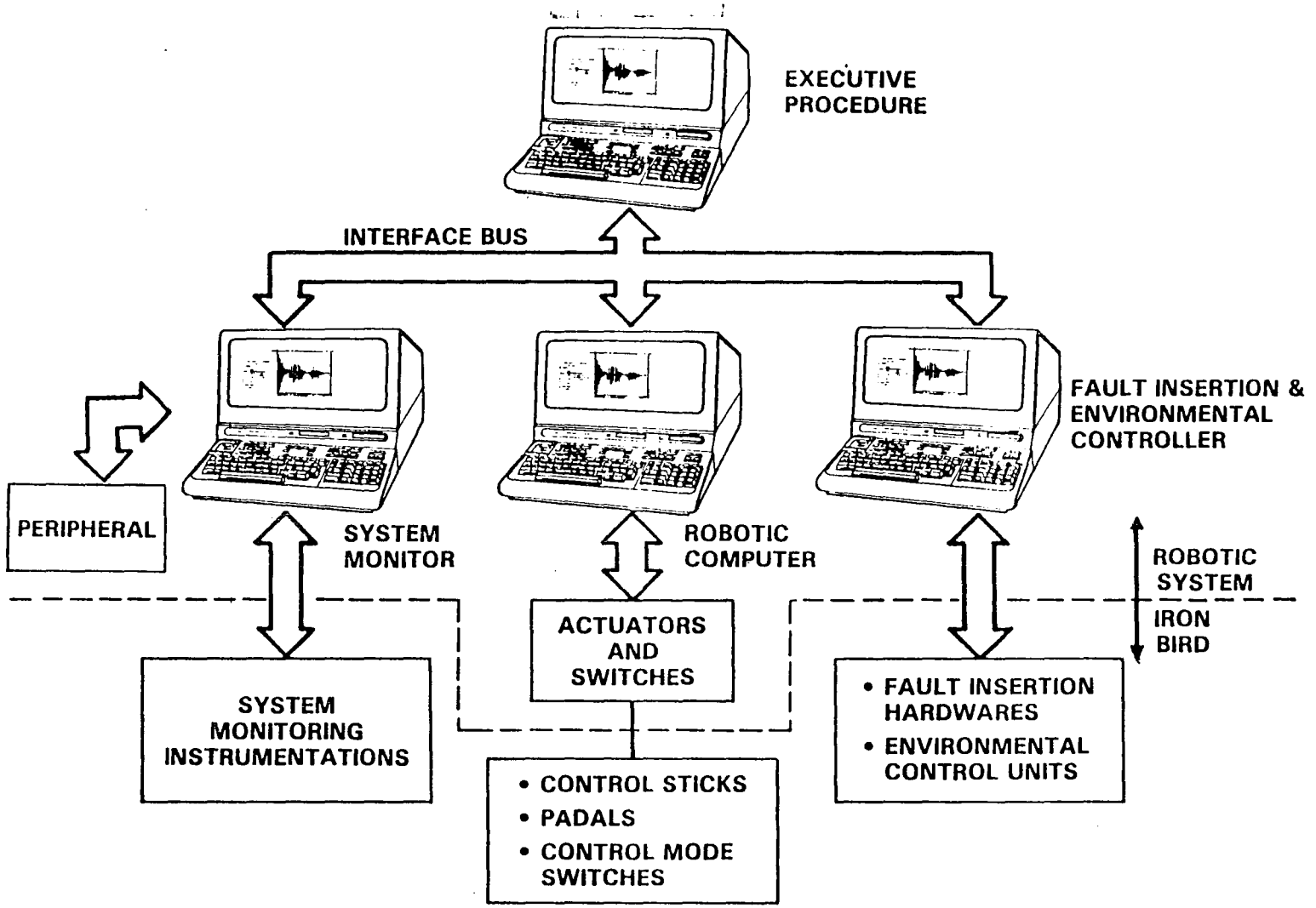


Figure 29. - Robotic system interface.

procedure module as the central controller. Other modules perform a specific programmed task without exchanging information with other units for total system operation.

This modularized approach offers the following advantages:

(1) The system can be easily modified for integration with other iron bird facilities.

(2) Modification and development of a module does not affect total system operation, therefore reducing down times and other costly delays.

(3) Other functions can be easily added to the system if determined necessary later in the program.

6.4 FLIGHT TESTING

The S-3A is well-suited as a testbed to develop FBW concepts. There are four crew stations--two of these are pilot flight stations and the other two could be used for special purposes; that is, fault injection panel, configuration switching, or onboard data processing. Each station has a full escape system including ejection seat, oxygen supply, and suit pressurization. The modest fuel consumption allows the plane to fly more than six hours. However, each wing-mounted engine has enough reserve power to fly aircraft individually. The highly maneuverable airframe was designed to withstand 3.5g maneuvers and accomplish takeoffs and landings in short distances. When the antisubmarine warfare equipment is removed, the existing avionics racks have sufficient room to install FBW avionics and extensive instrumentation electronics. The basic S-3A was designed for continuous duty with minimum maintenance.

Flight testing to develop FBW systems can be divided into two categories. The first is fault-free testing to verify the control laws, demonstrate handling qualities, and accomplish the usual envelope expansion. The second involves inserting faults in flight to validate fault tolerance.

Prior to flight testing, extensive ground checks will be made to ensure that the control system conforms to design specifications and iron bird tests. These checks will include end-to-end gains, system operational checks, frequency response tests, and ground shake vibration tests.

6.4.1 Performance Verification Flight Testing

The purpose of performance verification flight testing is to verify control law design and demonstrate that the handling qualities of the FBW system satisfy requirements. The basic FBW modes to be tested include the augmentation modes and the alpha limiting mode. All autopilot modes must also be engaged to demonstrate that there is no adverse interaction between the autopilot and the basic flight control system. The nominal loop gain can be verified to show stability margins. Finally, the aircraft will be flown in turbulence to assess gust responses.

6.4.2 Failure Mode Flight Testing

The purpose of these tests is to validate the fault-tolerant design and verify the fail-operational capabilities of the FBW control systems. A few selected tests of this nature will be performed to back up the comprehensive testing that was conducted on the iron bird.

There are two types of failure mode tests. The first simulates a system failure by turning off one or more channels of a major subsystem; that is, hydraulic #1, B electrical bus, no. 1 air data system, or no. 3 aileron actuator. The second type of failure test intercepts a signal going to the computer and inserts a bogus signal. This type of testing will be performed on the sensors, actuators, and the air data computer.

6.4.3 Flight Test Support

Special instrumentation and software modules will be used to support flight test. The purpose of the flight test instrumentation is to:

(1) Provide test data that validates FBW system performance

(2) Document all actual or simulated failures that occur

(3) Generate the engineering data necessary to enhance system design

(4) Provide real-time ground monitoring through telemetry

The onboard instrumentation system will monitor all record and telemeter commands, input signals, and the state of the computer at all times. Information generated will be available to onboard personnel and will also be transmitted to the telemetry station.

Software modules need to be added to the FBW computers for flight testing. Typical modules include the stability module, fault insertion module, and instrumentation module. The stability module will give onboard personnel the capability of changing system gain or time constants with external switches placed in the flight station. The fault insertion module will be used to verify the fault-tolerance capability of the software. The instrumentation module will transmit the state of the system and other pertinent data to the instrumentation system.

SECTION 7--S-3A INTERFACES

7.1 FLIGHT CONTROL SYSTEM INTEGRATION

The primary goal during this phase of the integration study effort was to determine a method of integrating the system into the S-3A aircraft with minimum cost. This would be accomplished by minimizing components developed specifically for the S-3A which would not particularly demonstrate features of the ADFBW system. The integration also should provide a flexible test vehicle with expansion capability and, above all, should assure flight safety.

The result of this effort suggested a method of integrating the ADFBW system with very little risk, development activity, design activity, or fabrication effort. The method is to use the existing surface actuators, eliminating all existing mechanical linkages and cables.

The baseline S-3A flight control system was originally designed with the objective that the aircraft be controllable even in the event of total loss of hydraulic and electrical power. This objective influenced almost every aspect of the design, even to the extent, for example, of using low-friction seals on the main ram of each surface actuator to reduce the force necessary to move the surface in the unpowered mode. This design feature was provided at the cost of lower seal life expectancy and lower reliability. Figures 30 through 32 show a schematic of the baseline S-3A primary flight control system in the pitch, yaw, and roll axes, respectively.

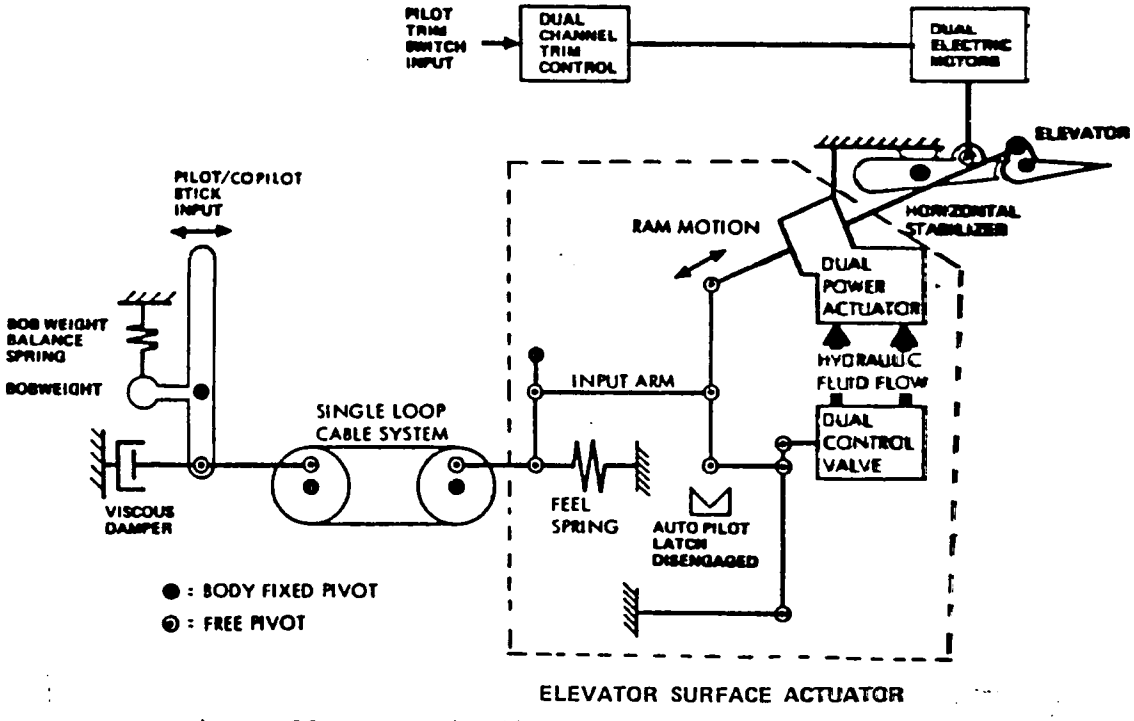


Figure 30. - Longitudinal primary control system.

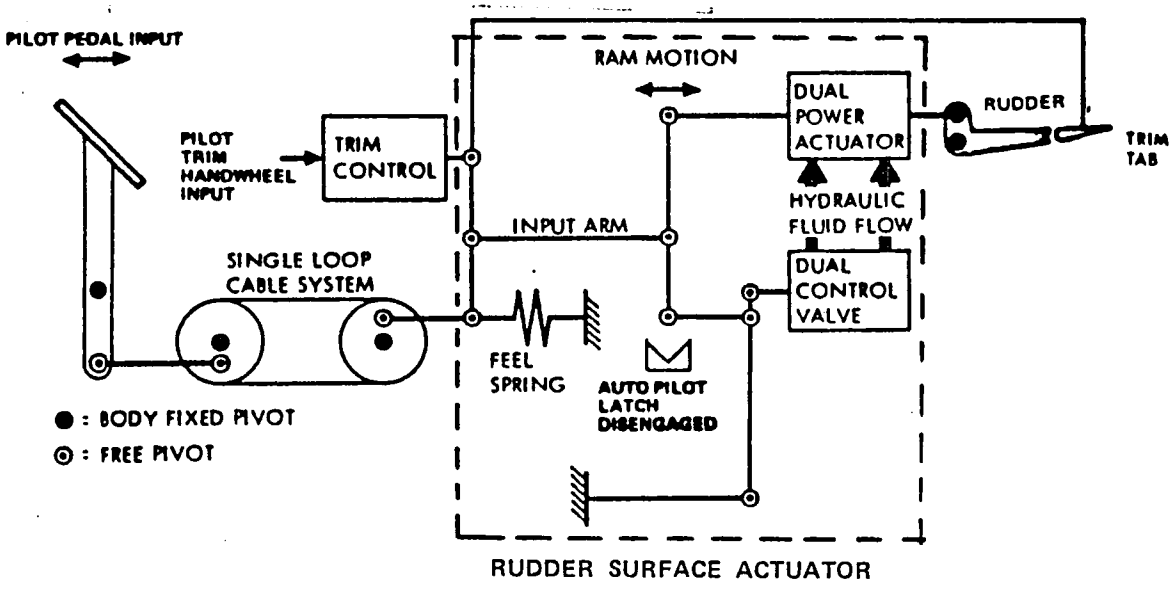


Figure 31. - Directional primary control system.

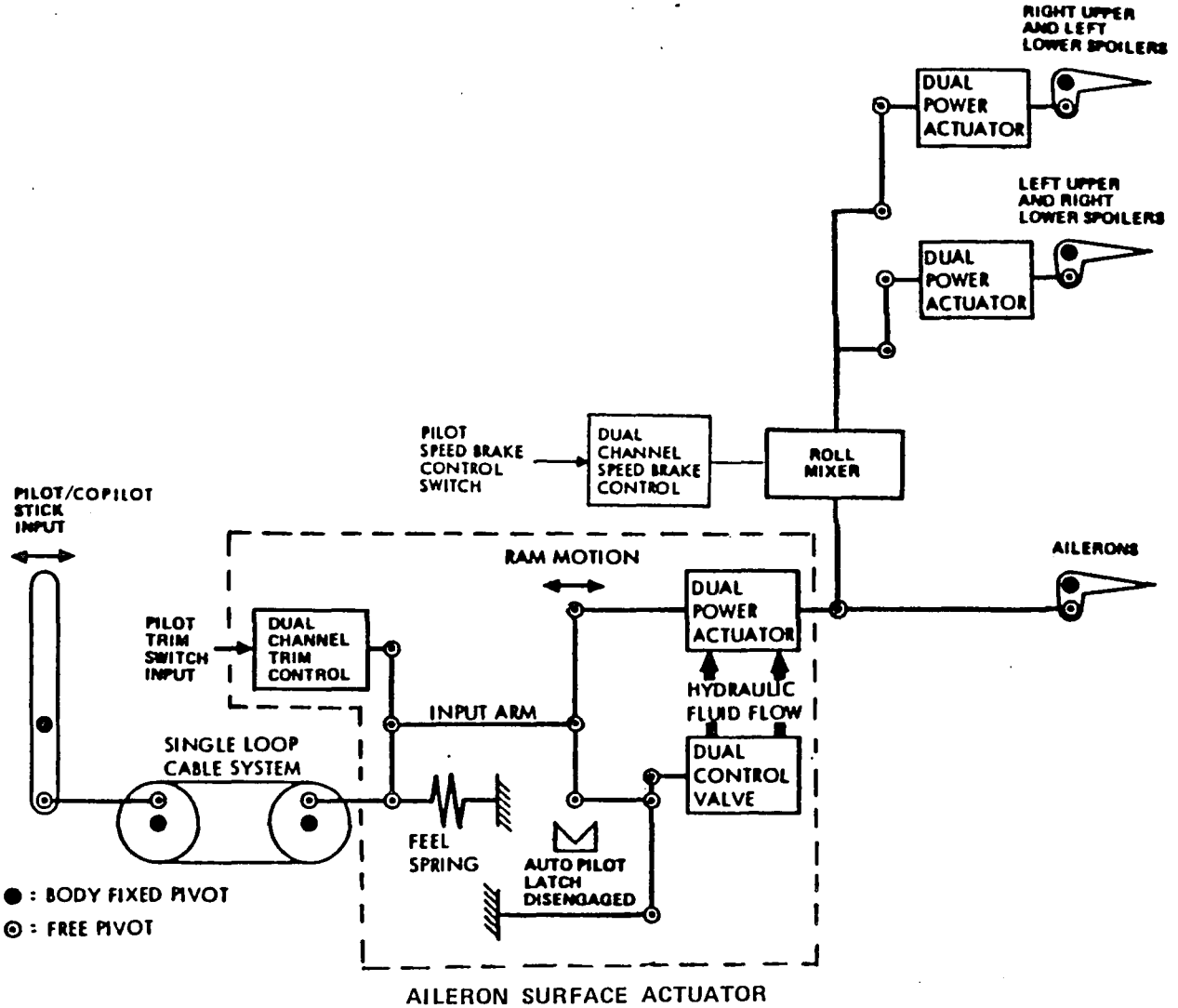


Figure 32. - Lateral primary control system.

Modifying the baseline design to accommodate a different objective requires that consideration be given to flight safety as well as to the cost of making the modification.

The modification we are proposing satisfies both flight safety and minimum cost. It consists of

(1) Completely decoupling the cockpit controls from the flight control system

(2) Using the existing cockpit controls for inputs to the ADFBW system

(3) Using only the control surface actuators from the existing flight control system

A preliminary layout was made to investigate the feasibility of installing secondary actuators in the pitch, roll, and yaw axes. The layout shows the installation to be feasible with no major rework of the aircraft necessary.

The addition of the secondary actuator introduces an additional time delay in the control system that does not exist in the basic S-3A flight control system. If the secondary actuator is designed to have high-frequency response, the time delay will be small and is not expected to significantly degrade aircraft handling qualities.

The ability to use existing S-3A surface actuators at the beginning of the test program allows a most significant economy. Designing, testing, and demonstrating that a new actuator meets hinge moment, stiffness, frequency response, hysteresis, and flutter requirements is not required because the existing actuators already provide these capabilities. Furthermore, new actuators may present a maintenance problem (repairing leaking servos, etc), whereas replacement spares for the time-proven original actuator, if required, are probably available from Navy inventories.

It appears that the same type of secondary actuator can be used for all three axes.

A preliminary sizing of the secondary actuator was made for the purpose of determining installation space. The force requirement for the secondary actuator is predicted on

(1) Operating the surface actuators in the manual mode--that is, unpowered

(2) Retaining the feel cams from the surface actuator

The secondary actuators should provide a force input to the primary surface actuators equivalent to that provided by a typical pilot acting on the controls in the cockpit of the basic S-3A. The feel cams could be removed from the primary surface actuator, thereby reducing the force requirements of the secondary actuators. However, in order to remove the feel cams, the actuator would need to be disassembled at the factory and then retested after rework. The actuators would become unique and repair or replacement more difficult.

The force and stroke required of the secondary actuators operating in a single-channel mode is 500 pounds and 3.0 inches.

A Hydraulic Research secondary actuator, which is triple-channel tandem with only one active channel operating at a time, was selected for this application. This actuator has a size of about 5.5 inches x 4.5 inches x 16.5 inches. This actuator provides 900 pounds force output with a 1.0-inch stroke. A slightly larger piston diameter and a bell crank to change gearing to the correct stroke will allow this actuator to meet requirements.

A preliminary layout was made of the installation of the selected secondary actuator driving the elevator power servo. Figure 33 shows the result of the layout with the secondary actuator positioned just forward of the primary servo. The tension regulator is removed to make the space available. The selected installation has reasonable access by way of the access hatch located in the left landing gear well, which leads into the environmental control system compartment.

A similar preliminary layout was made of the yaw axis secondary actuator installation. As shown in figure 34, adequate space exists above the fin fold line to install the actuator near the rudder primary actuator. The actuator is shown mounted on the forward spar. Access to the installation will be available from the rudder servo access plate on the right side of the fin above the stabilizer.

The aileron secondary actuator installation presents a more difficult installation. The result of the preliminary layout is shown in figure 35.

Access to the aileron secondary actuator installation will be the access door to the environmental control system compartment.

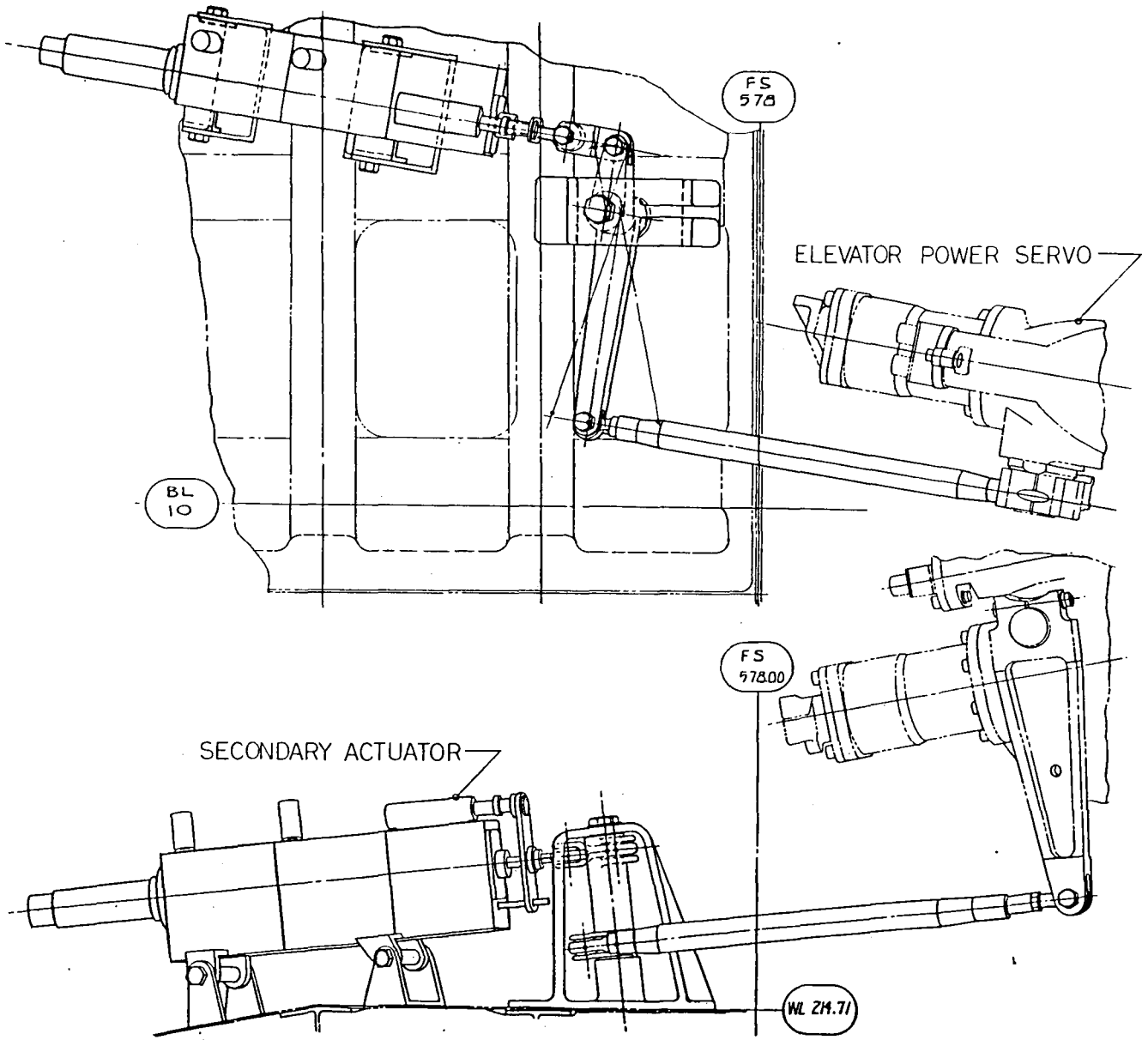


Figure 33. - Elevator secondary actuator installation.

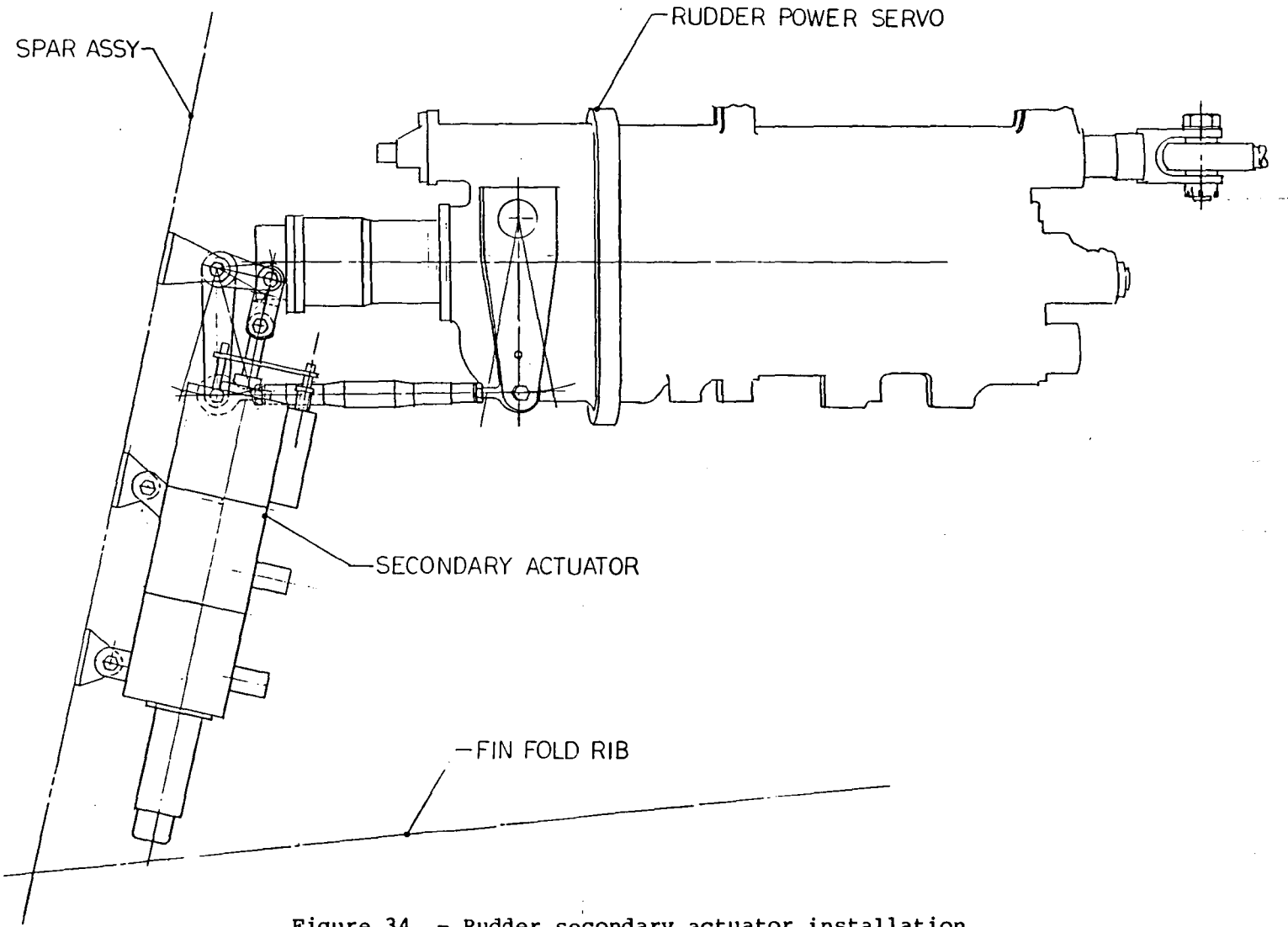


Figure 34. - Rudder secondary actuator installation.

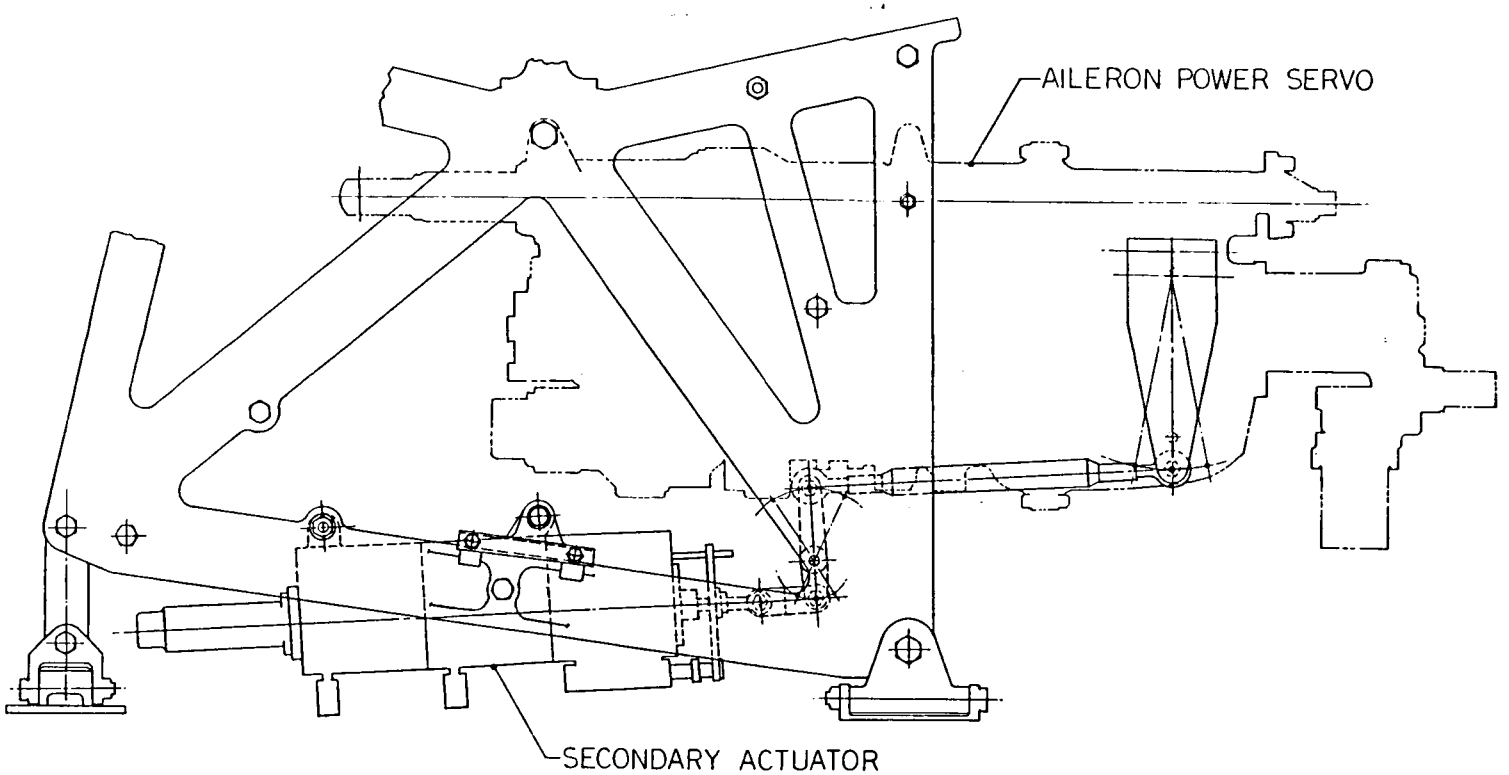


Figure 35. - Aileron secondary actuator installation.

Separate new actuators are required to drive the ailerons in a symmetric manner. This can be achieved by using two additional series actuators, one for each aileron. This configuration was selected because of the following considerations. The ailerons are controlled by a single aileron power servo, installed in the environmental control system compartment, which drives pushrods to each aileron. The aileron servo also provides a mechanical input to the spoiler actuator. Rather than redesign this total system, it is recommended that additional actuators be used.

The actuators selected for the symmetric aileron mode will be electromechanical actuators. Development of the mode could be a follow-on expansion program after the flight test of the ADFBW system has commenced.

The logical location for the electromechanical actuators is in each wing because they must work in conjunction with the aileron pushrods, and space in the fuselage between the aileron actuator and the aileron pushrods going out of the fuselage is very limited. The space available in the S-3A wing is limited because of fuel tanks and hinges for folding.

A brief effort was made to determine the feasibility of locating the actuator in the wing. Figure 36 shows the result of this effort. The actuator was sized for the available space at this location instead of for the task required. It is likely the space is too small for an electromechanical actuator. The installation is made practical by removing the lower spoiler dwell actuator. In so doing, the lower spoilers are made inoperative and the panels are sealed closed. The baseline S-3A uses the lower spoilers only when the flaps are up. The loss of roll control power will not be significant. Some slight pitching movement may result at high-speed roll maneuvers due to not using the lower spoilers and only using the upper spoilers. This should not present a problem because with the ADFBW a simple crossfeed of roll into pitch can significantly reduce any pitching moment.

The feasibility of alternate installation areas should be determined. Another possible actuator location is in the wing beyond the wing fold line. A third possibility is in the environmental control system compartment near the aileron primary and secondary actuators. Future study should be devoted to these areas.

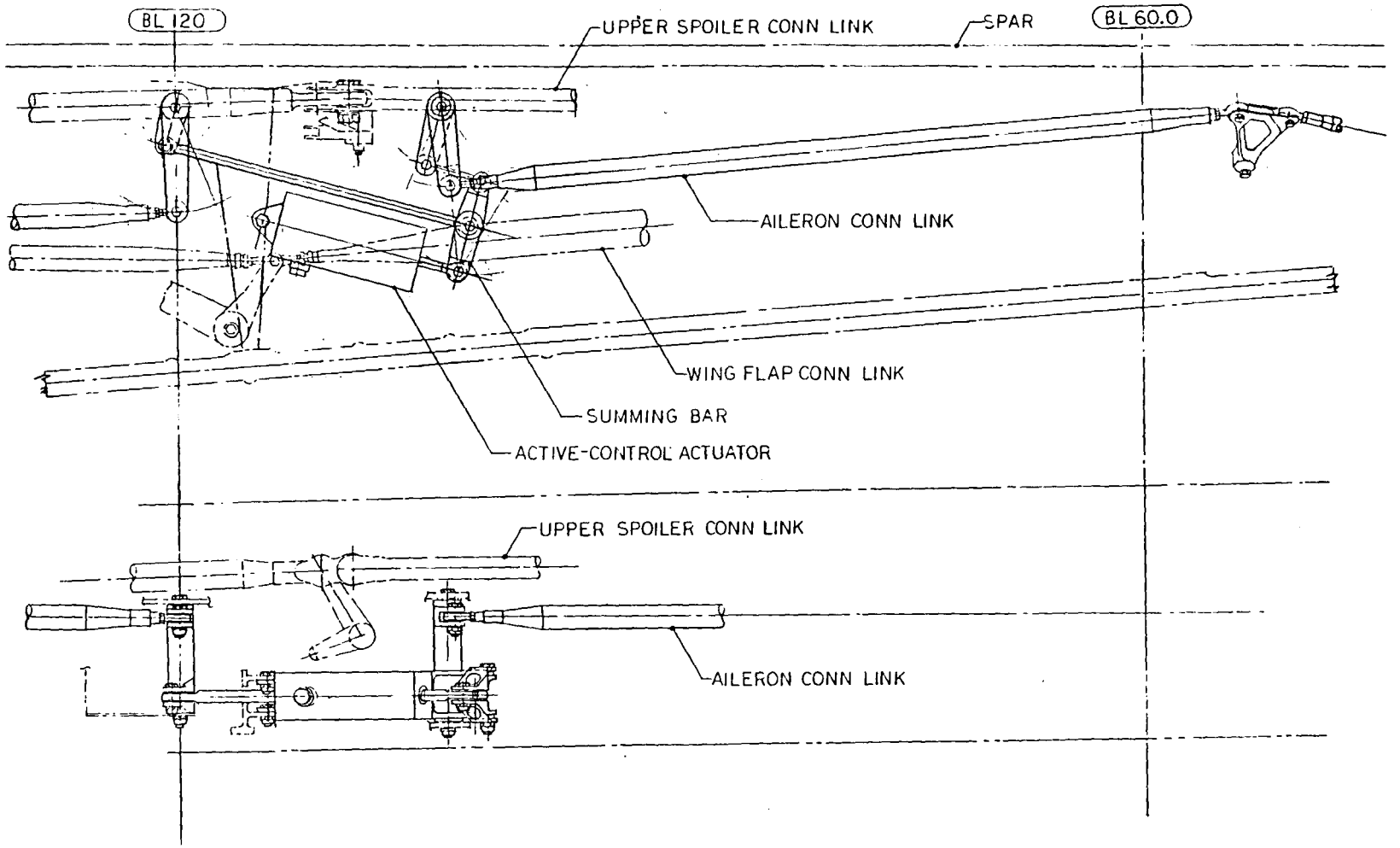


Figure 36. - Active-control actuator installation.

In summary, the locations of all the flight control actuators are shown in figure 37.

7.2 COCKPIT CONTROLS INTEGRATION

The task of providing cockpit controls for the ADFBW requires compromise to avoid much redesign effort. If a new control stick or side-stick controller is to be provided, many of the additional functions provided by the original control stick and pedals must also be provided. Some of these functions are nose gear steering and its engage/disengage function, pitch and roll trim, trim disconnect, autothrust disengage, autopilot disengage, communications switch, and brakes.

A right-hand controller is required if other than a center stick is used because the throttles are located on the left side of both pilot and copilot. Mounting the controller would require a different installation for pilot side (center console) and copilot side (side console); thus, two designs are required. Both installations must avoid interference with the ejection seats.

Consideration of the above design tasks persuades us to recommend using the existing installation, with modifications as required. The modifications will consist of removing their respective cable systems, the bobweight, and the stick damper. The column should be statically rebalanced by means of the balance spring, a spring gradient added to provide a stick force gradient and pedal force gradient, and six LVDTs installed per axis to measure control inputs. No installation difficulties are foreseen with this approach. A diagram of the reconfigured controls is presented in figure 37.

7.3 SENSOR INTEGRATION

Individual sensors do not present an installation problem. However, installing sextuple sensors requires suitable space so that each of the sensors is in the same environment as the others to ensure that cross-channel monitoring can have practical levels. In some cases, such as the pitot-static system, this may be impractical.

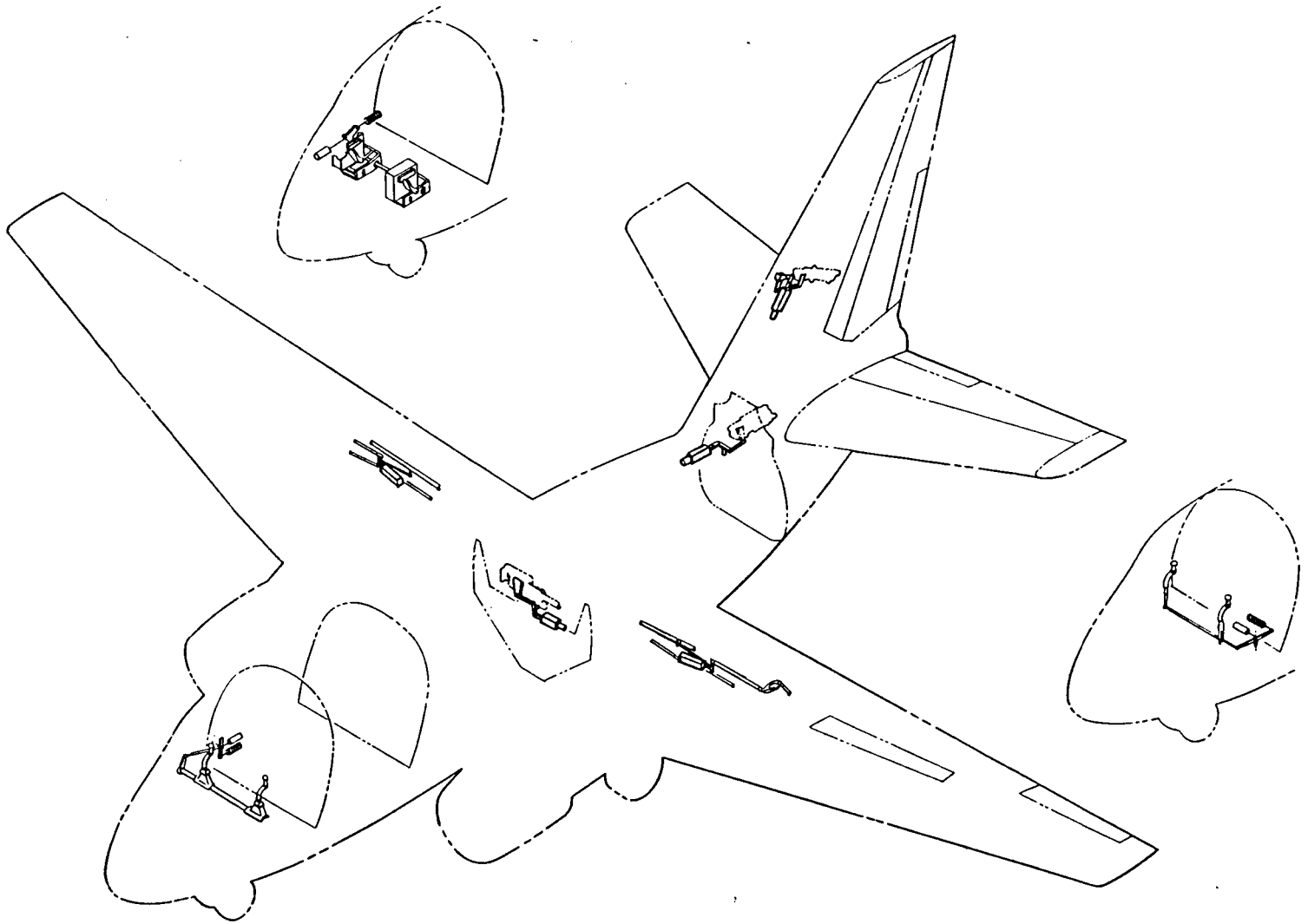


Figure 37. - Summary flight control changes.

Installing multiple pitot-static probes on a small aircraft presents more of a problem than on a wide-body aircraft. Separating the systems as widely as possible so that bird strikes will not damage all systems is not consistent with providing the same airflow around each probe for multiple-channel tracking.

The following configuration is recommended based on the history of probe development and probe installation on the S-3A. (See figure 38.)

The present two-channel pitot-static system installed on the S-3A should be extended to a four-channel system. The baseline S-3A has a probe on each side of the fuselage, ahead of the cockpit. Each probe has two static ports and a single pitot input. Each static port is crossfed to the opposite port on the other probe. In this manner effects of sideslip are minimized.

Probes identical to those presently installed on the S-3A should be specified so that correction for static defect curve (probe angle-of-attack effects) need not be reestablished. The correction will be particularly important for four-system tracking as required for cross-channel comparison.

Part numbers and the approximate location for the new probes are shown in figure 39. The new probes will be mounted just under the existing probes but with enough separation to minimize shadowing for most angles of attack. Additional lines must be plumbed from the new probes to the right internal electronics bay. Six air data computers will be coupled to the four pitot-static systems.

Rate gyros and accelerometers can be mounted in the bomb bay area on the keelson approximately at the wing quarter-chord. Adequate space exists for the installation.

7.4 AVIONICS INTEGRATION

Ample space and facilities exist on the S-3A aircraft for installing the new avionics of the ADFBW system by removing the unnecessary avionics. It should present no installation or access difficulty.

The internal avionics racks will provide sufficient space with hard-mounting possibility. The racks will have a controlled temperature environment and adequate cooling will be supplied by the ducted avionics cooling system.

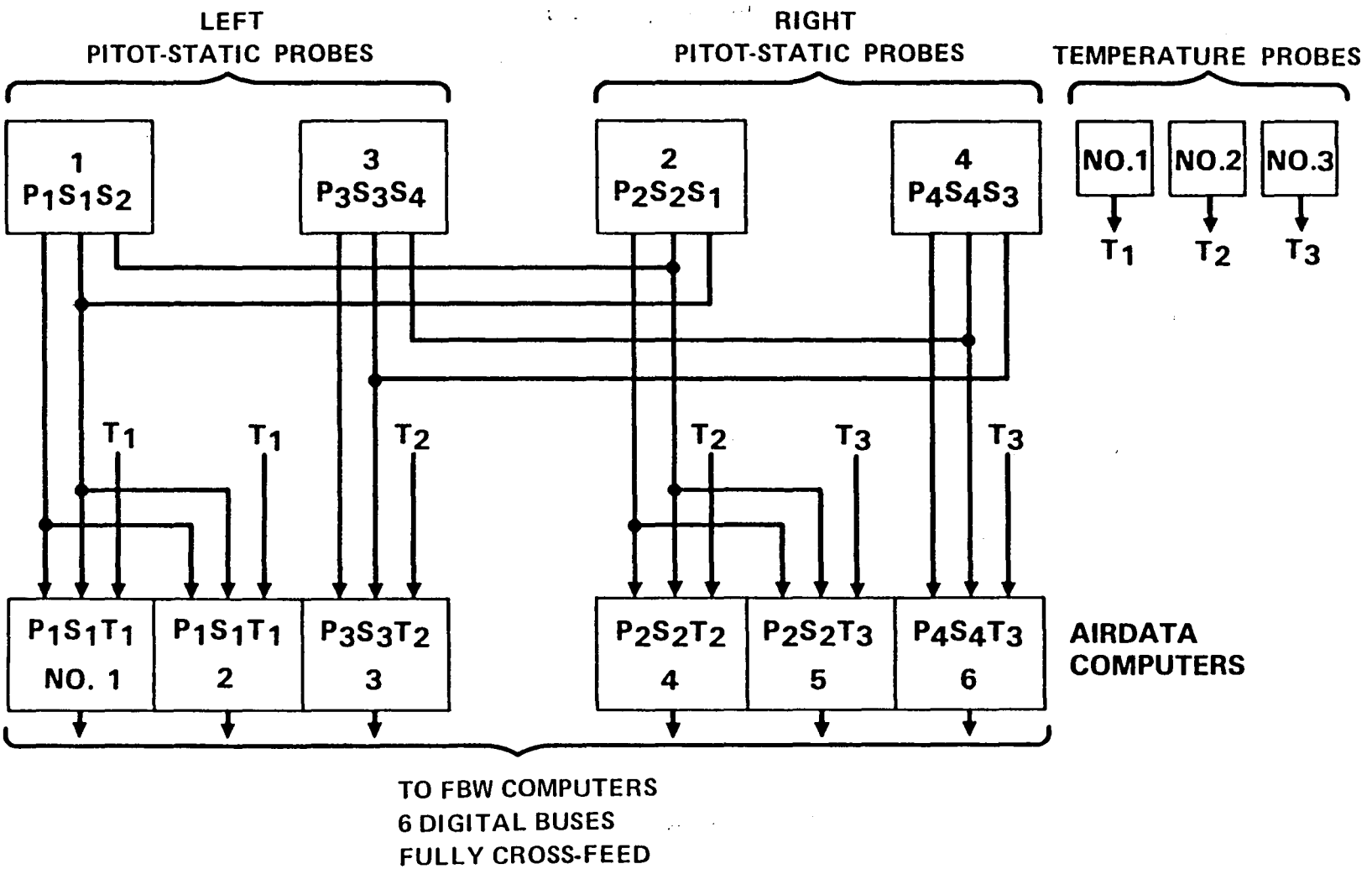


Figure 38. - Air data system interface.

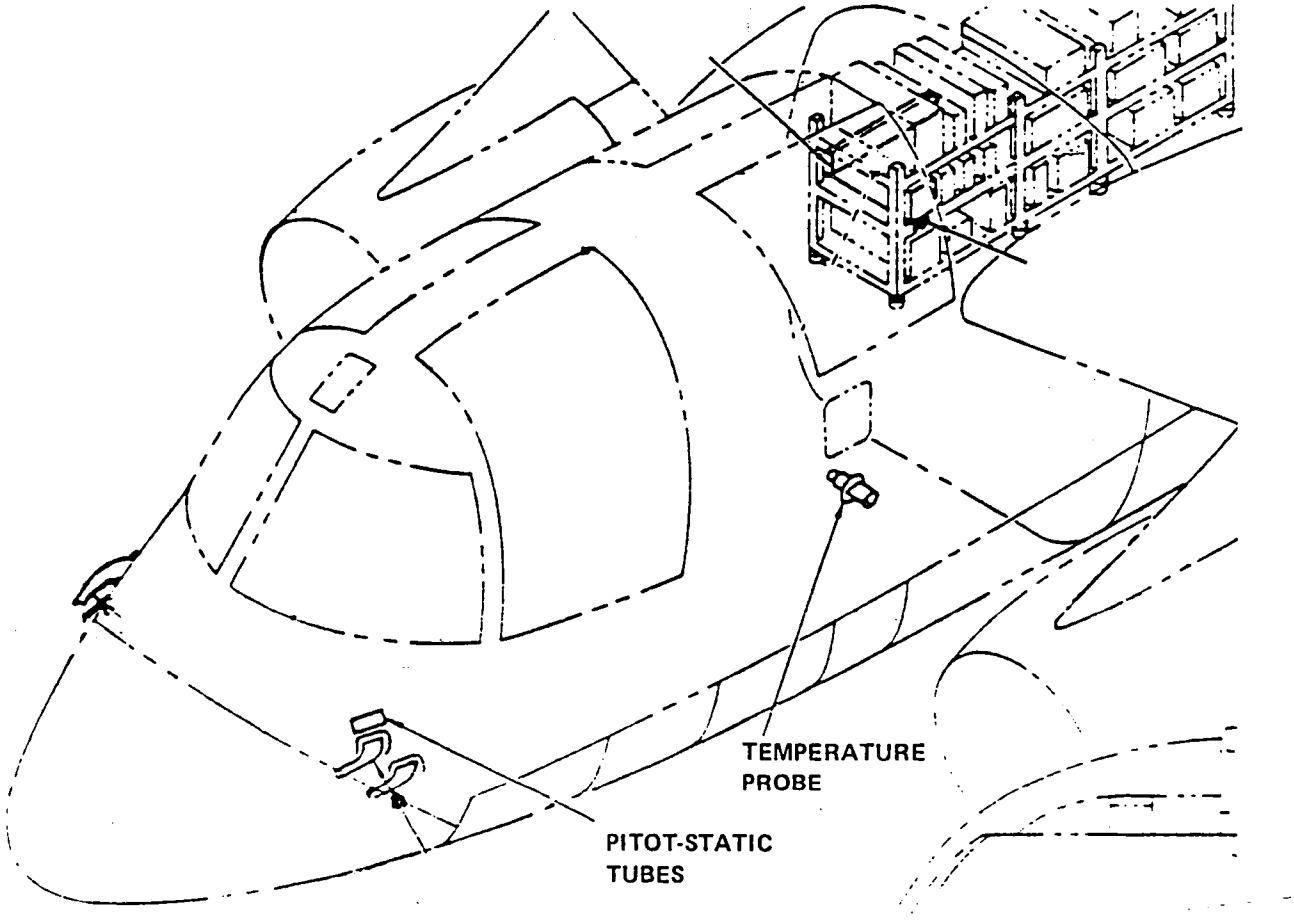


Figure 39. - Pitot-static system.

The two remote terminals can be installed in either a controlled or an uncontrolled environment area. The internal avionics area will provide environmental control. The environmental control system compartment, or the bomb bay area, will provide ample space for avionics in an uncontrolled environment. Both areas are accessible for ground checkout and maintenance.

7.5 HYDRAULIC SYSTEM

A review of the S-3A hydraulic system (including approved ECP 347 changes) was made regarding its adequacy to support the ADFBW system. Flight safety of the flight test vehicle was the paramount aspect of the review. It was concluded that no significant changes to the S-3A hydraulic system are required.

The baseline S-3A has two engine-driven, independent hydraulic systems to provide dual-channel flight control hydraulics. System 1, the flight control/utility (FC/U) system, powers all utility functions (landing gear, brakes, nose gear steering, flaps, etc) in addition to the flight controls. System 2, the flight control (FC) system, powers only the second channel of the flight control system. As shown in figure 40, each system can by itself provide complete flight control capability.

A third hydraulic pump will be added when ECP 347 is incorporated. This pump, driven electrically by either generator (but not by the baseline APU) will be plumbed into the FC/U system and will function as an emergency backup pump. It the left engine is shut down or the FC/U system pump fails, the emergency pump can be used to do all the work of the engine-driven pump.

Throughout the history of the S-3A as known by Lockheed project engineering, total hydraulic power has never been lost. (At this time, approximately 1% of the aircraft have the third hydraulic pump incorporated.) Based on this history, the baseline dual system appears adequate and safe to support the ADFBW system. Adding the third hydraulic pump should reduce the chance of losing all hydraulic power even further. In addition, the third hydraulic system will provide hydraulic ground checkout capability without needing a hydraulic rig. Only an electrical ground cart is required.

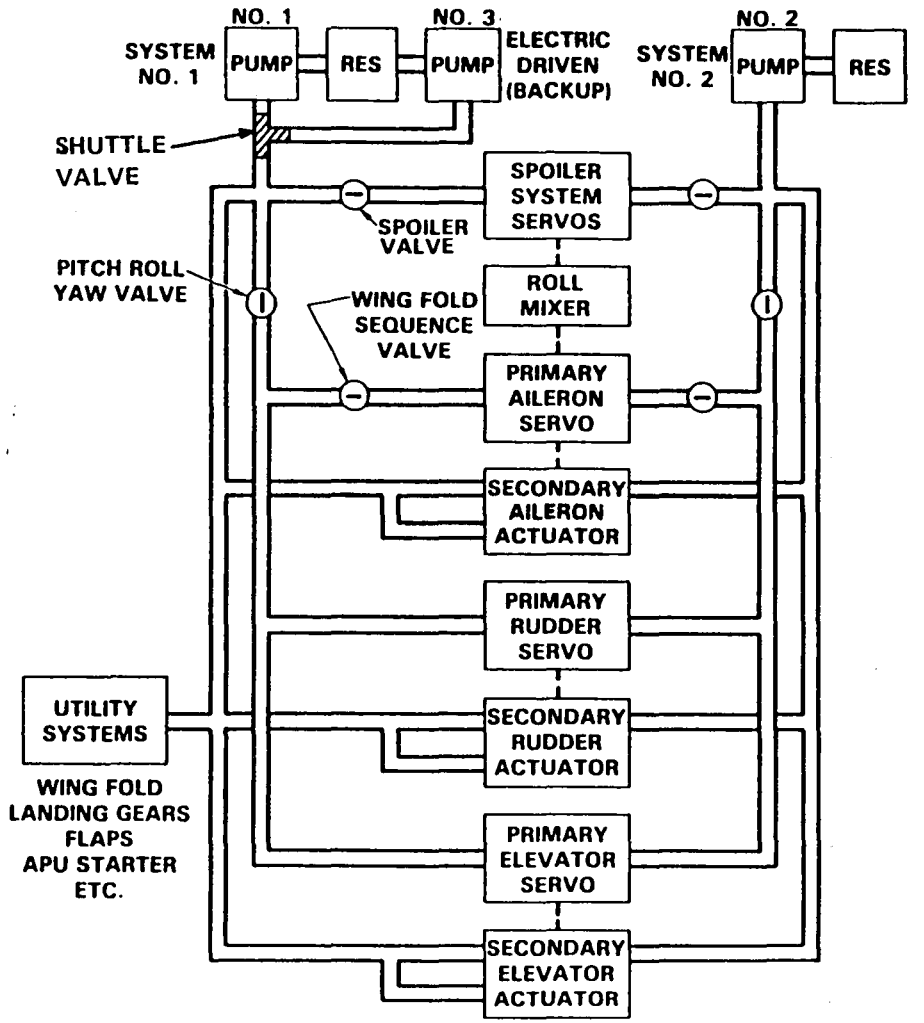


Figure 40. - Hydraulic distribution system (dual-channel).

The modification of the existing hydraulic system will consist of teeing into the pressure and return lines of both the FC/U and FC systems. The tee will provide hydraulics for the pitch, roll, and yaw secondary actuators. Two channels of the secondary actuators will be powered by the FC/U system and one channel will be powered by the FC system.

7.6 ELECTRICAL SYSTEM

The S-3A electrical system has been reviewed with respect to the anticipated requirements imposed on it by the ADFBW system. It is concluded that the impact is minimal and no extensive or costly modification to the S-3A electrical system will be required.

The baseline S-3A has two separate engine-driven generators, each of which can be bused to supply all electrical needs. An additional 5 kva generator is powered by the APU. This unit is only large enough to supply essential electrical power, such as the pitch trim actuator and flight instruments (refer to figure 41).

An engineering change proposal (ECP) is being prepared to install a larger APU in the S-3A. This unit would be capable of supplying the ADFBW system. An S-3A with the ECP incorporatd (larger APU) should be used for the ADFBW demonstration aircraft.

A 28V dc storage battery sufficient to supply electrical power to two of the six channels of the flight control system will be installed in the bomb bay compartment. Two typical 12V batteries should be adequate to power two channels of flight controls for at least two hours in case of an emergency.

SECTION 8--CONCLUSIONS AND RECOMMENDATIONS

This study has defined an architecture and a methodology for its development. The architecture claims advantages by virtue of

(1) Less software

(2) Self-checking hardware

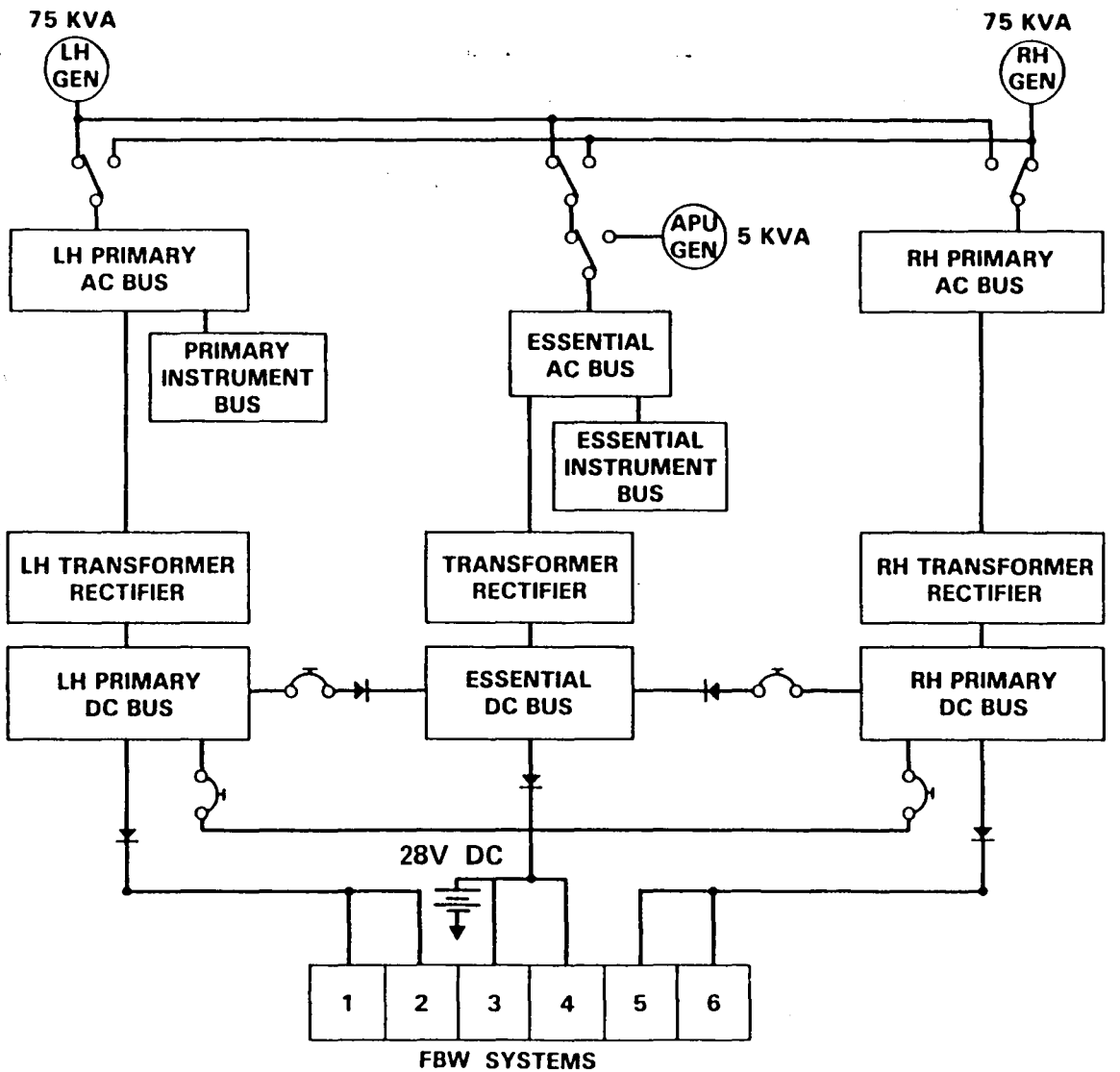


Figure 41. - Electrical power system.

(3) Emphasis on using standards

Instruction set:	1750A
HOL:	Ada
Serial bus	1553B

(4) Hardware advances that yield maintenance benefits by including spare elements

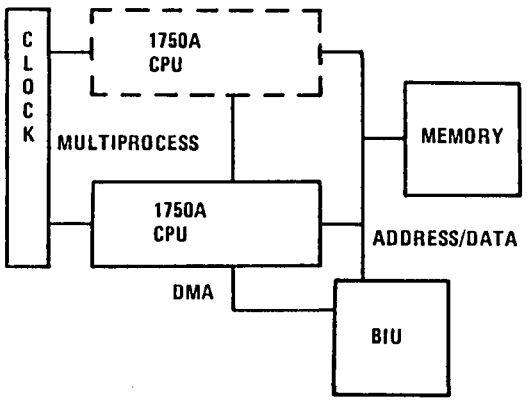
This architecture has emphasized simplicity. This yields tractable modeling problems to estimate reliability and easier verification of the software. In spite of the hardware-intensive nature of this architecture, we recognize that software requirements will grow as new functions are integrated with flight control. Two options for expanding the computer throughput are shown in figure 42. Figure 42a shows the addition of another CPU chip using the multiprocessor option provided on the Fairchild 1750A. This feature allows both processors to access a common memory without contention problems. Figure 42b shows the addition of SCMPs to the redundant sensor bus. These additional processors can perform non-flight control functions (i.e., navigation/flight management). Spare SCMPs may be used as back-ups for both flight control and non-flight control functions. Investigation of functions other than flight control was beyond the scope of this study. However, this architecture is well-suited to expansion.

The methodology claims advances in

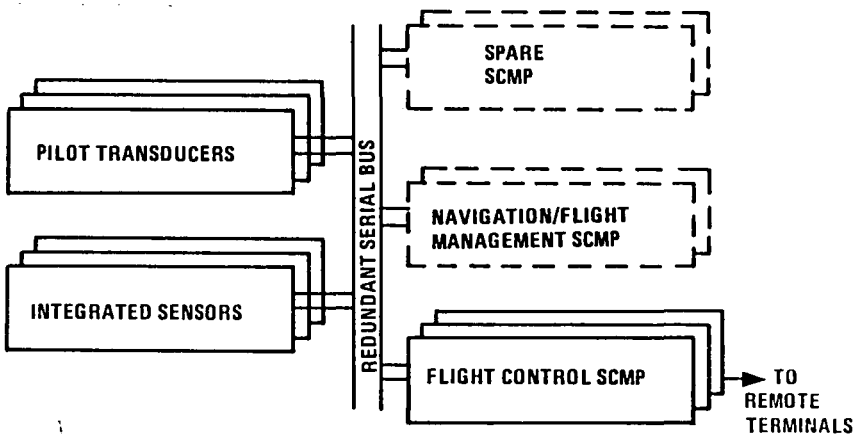
(1) Presenting aids to formulate complete specifications

(2) Using finite-state descriptions and fault tree models to define test cases

The development methodology proposed is within the state-of-the-art, and is a cost-effective way to produce flight-critical software. Research in fault-tolerant computing is currently an active area. Work is ongoing in many areas, including reliability modeling, fault-tolerant software approaches, formal methods for design proving, and design of real-time operating systems. As previously noted, software requirements are increasing and may benefit from the results of this research. We view the entire development methodology as continually evolving and expect to incorporate new techniques as our knowledge expands.



a. Tightly coupled multiprocessing



b. Adding SCMPs to the bus

Figure 42. Options for computer expansion.

Finally, automation of iron bird testing is recommended to achieve time compression and enhance productivity. We believe this is a good time to initiate the ADFBW development. Figure 43 suggests follow-on activities.

8.1 RECOMMENDED FOLLOW-ON

Three one-year efforts are shown in parallel to support the major development activities in phase 1.

The system specification activities define the technologies used to implement the elements of the architecture and detail the interfaces. The system specification will be sufficiently detailed to permit hardware design to start in phase 1. A top-level system specification is prepared and the fault tree reliability modeling is expanded.

The test design methodology addresses issues raised in section 6 concerning the application of finite-state machines and fault trees. These techniques will be examined and results extended to evolve a practical methodology for designing test cases.

The robotic demonstration is intended to take an initial look at the issues of automating iron bird testing. It develops the computer interface, robotic actuators, and system instrumentation. A demonstration on an existing facility (like the F-8) is recommended.

Following these support activities a two-phase program is shown, leading to and including flight test. Phase 1 comprises those tasks required to complete iron bird testing of the ADFBW system. This phase lasts three years and includes:

(1) Detailed hardware design

(2) Software design and coding

(3) Interface checkout, system simulation, and accelerated life testing

(4) Iron bird testing

Development of verification and validation tools occurs in parallel. This activity designs the test cases and develops the hardware and software necessary for automated iron bird testing.

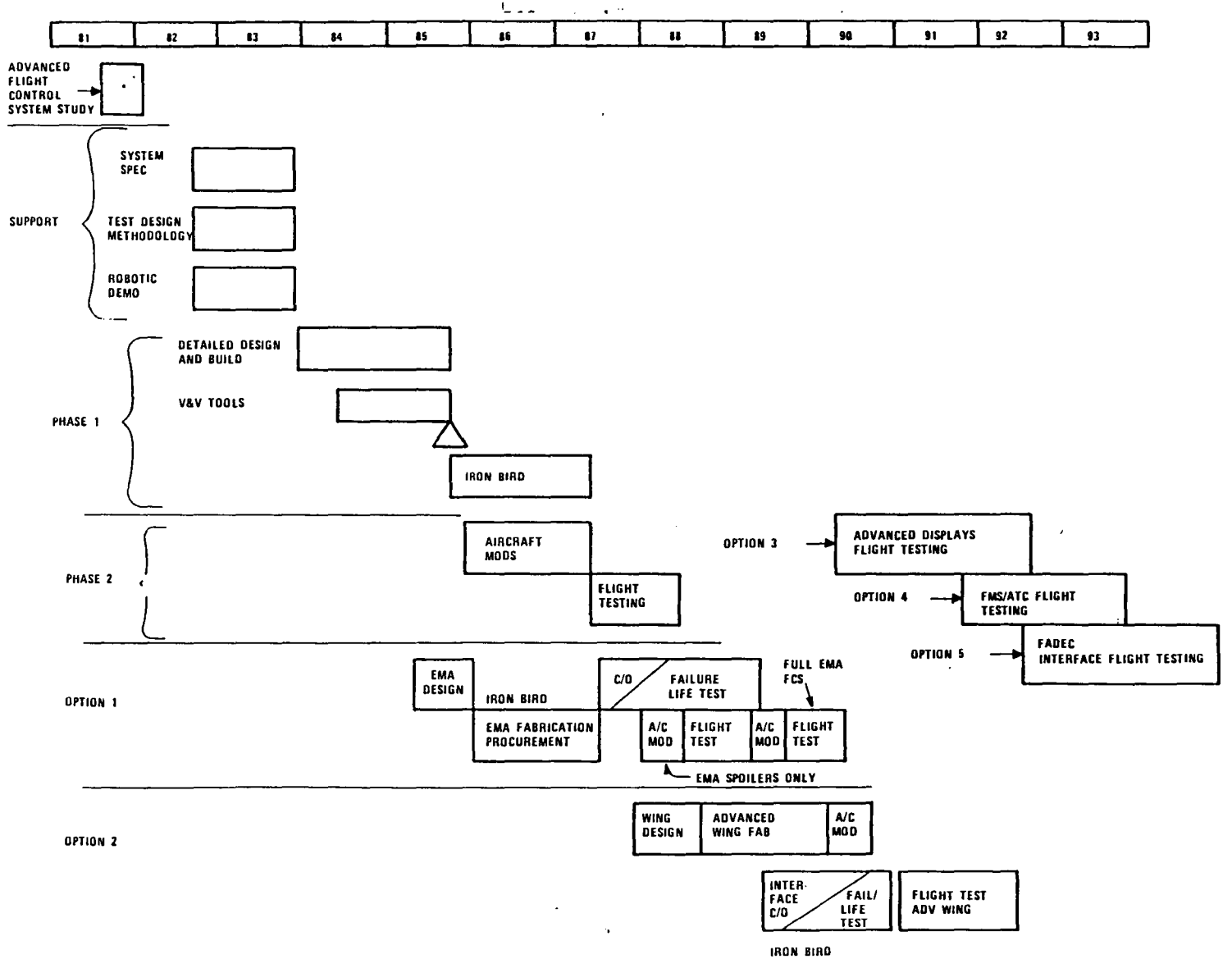


Figure 43. - Recommended follow-on activities with options.

Phase 2 is the flight test phase for the basic ADFBW sytem. It includes aircraft modification and one year of flight testing to verify the performance of the basic design. It could start in mid-1987.

The S-3A ADFBW test vehicle could support a variety of flight research experiments. The five options shown in figure 42 are described below.

Option 1 involves testing of electromechanical actuators. Tasks are proposed in a time frame that resumes flight testing in the shortest time following the basic phase 2 flight test interval. The ADFBW system will follow the verification and iron bird test methodology developed and used in phase 1. Flight test follows, conservatively at first, with the electromechanical actuators used only for powering the spoilers. The electromechanical actuators' successful service in a non-flight-critical application establishes their readiness for flight testing in all the surface actuator positions.

Option 2 flight tests an advanced wing requiring extensive use of active control technology. This effort could start in 1989 or earlier, depending on other research and development activities in this area.

Option 3 involves using advanced displays. These flight tests are scheduled for 1988-1989, at which time flat-panel devices should be available. A low-risk look at this technology can be made with an early flat-panel test in a rear crew position of the S-3A. In fact, years of "ridealong" testing could be acquired in a very nonobtrusive fashion in the rear position. When adequate reliability is achieved, one or both forward positions could be equipped with the advanced displays.

Option 4 is a test of flight management system and air traffic control system integration possibilities, rather than an application of ultra-reliable electronic technology. Rated as a fairly low-priority option it could, however, provide the S-3A test vehicle with the necessary avionics to engage in sophisticated air traffic control experiments. An example would be multiple airplane tests of curved approaches to microwave landing system airports.

Option 5 adds the capability to operate one engine with digital engine controls. The S-3A is an ideal testbed for this purpose since it has two engines and excellent performance, even with one engine out. In the proposed

study the flight propulsion control coupling possibilities and engine/airplane electric power sharing will be investigated.

The preceding discussion illustrates that the ADFBW S-3A would result in a flexible testbed vehicle for NASA that could support a variety of research areas. This theme is expanded on below.

8.2 EXPANSION TECHNOLOGIES

The preceding sections have described an ADFBW system. We have discussed the issues involved in installing such a system on the proposed testbed aircraft on S-3A. We conclude this technical discussion with a brief look at several "expansion technologies" that could be the subject of research and development at NASA on the S-3A ADFBW aircraft. These expansion technologies are non-flight-control technologies that will figure prominently in the efficient all-electric aircraft of the future. Figure 44 is a roadmap of five expansion technology areas. The development of these technologies over the next 20 years is outlined below.

8.2.1 Active Control and Advanced Wing Technology

Active control technology is represented along the bottom portion of figure 44. New aft-loaded airfoils require CG placement that yields relaxed static stability to the point that FBW stability augmentation is needed. Next-generation wing designs will require FBW technology to achieve the optimum aerodynamic and structural efficiencies.

Using the S-3A as a test vehicle could be valuable in this area. A new wing could be demonstrated. On the new wing, active controls would provide maneuver load control, in which symmetric aileron deflection would unload the tips at load factors greater than 1g, thus conserving wing structural weight. The pitch control system, acting in cooperation with the ailerons, would provide elastic mode suppression and gust alleviation. Relaxed static stability would be used to get the most efficiency out of the advanced airfoil. Active CG management could be included to maximize fuel savings payoffs. Testing of the advanced wing with the S-3A ADFBW could take place in the late 1980's.

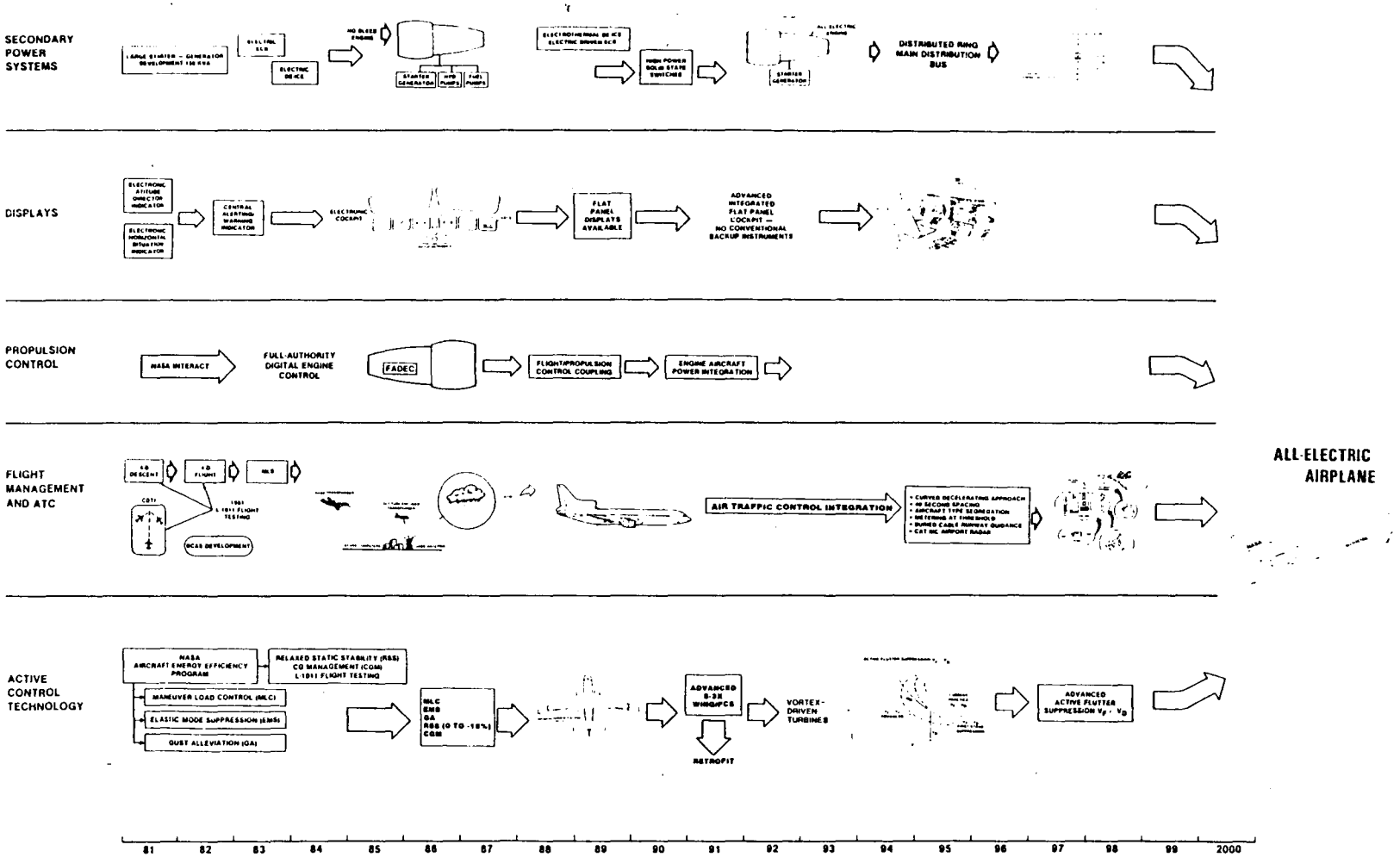


Figure 44. - Expansion technology flow chart.

Two technologies that can be integrated with an advanced wing are the vortex-driven turbine and upper-surface blowing concepts. The vortex-driven turbine can be installed for vortex dissipation and power extraction. Upper surface blowing is a technology for vectoring the thrust to increase circulation lift.

Active flutter suppression is seen as the last active control entry because of its rapid and drastic failure modes. A first test application might be with flutter margin reduced to dive speed; then subsequent flutter margins could be eliminated to the point where flutter margins could be provided completely automatically.

8.2.2 Flight Management and Air Traffic Control Technology

Flight management and air traffic control technology are combined in figure 44. Today the component pieces of the future air traffic control system are being developed. In general, airborne avionics are leading the ground-based air traffic control system toward an era of fully automated, high-density flow management. As flight control becomes more automated the crew will function increasingly as system managers. The primary emphasis will be on efficient and safe flight in a crowded traffic environment, with elimination of delay and with optimal accommodation of changing weather situations.

8.2.3 Propulsion Control Technology

Propulsion control as shown in figure 44 is heading inexorably toward full-authority digital engine control configurations. Digital engine control will provide opportunities for improved fuel efficiency and for the coupling of flight control and aircraft power systems. Gains for the relatively simple subsonic fanjet engine control will not be as impressive as those projected for transonic applications.

8.2.4 Display Technology

Electronic display technology will develop rapidly in the 1980s, culminating as shown in figure 44 with fully integrated, solid state flat-panel displays. The data volume available to the crew is almost

overwhelming today and will worsen with more widespread CRT usage in the cockpit of the near future. Much remains to be done in the area of human factors. NASA, recognizing this need, has initiated a substantial research and development effort in this area. Equally important, however, is the need for FBW-quality reliability in the displayed information. The advanced cockpit of the 1990's will be totally electronic. Its reliability must be equivalent to the electronic flight control because, as is the case with advanced flight controls, total loss of the displays could result in loss of the aircraft. Display technology then can directly benefit from ADFBW research efforts toward developing ultra-reliable digital equipment.

8.2.5 Secondary Power Technology

Secondary power systems will evolve in the 1980's and 1990's toward an all-electric implementation. Today's combination of electric, hydraulic, and pneumatic power has led to a proliferation of power sources and distribution systems, with a comparatively limited capability for load or function sharing. A two-phase evolution toward all-electric secondary power is anticipated. In the first, bleed power and pneumatic start systems will be eliminated. Engine starting and environmental control system power will be provided from a scaled-up electric power system. The second phase is more difficult; it involves the elimination of hydraulic power. All hydraulic motors and actuators will be replaced by electric-power devices. Hydraulic technology of today is as reliable as the structure of an aircraft. Many commercial transports are totally dependent on hydraulics for powering primary flight controls and other important systems. Emergent electromechanical actuators will have to be capable of equivalent reliability to replace hydraulic actuators in flight-critical applications.

The ADFBW program plays an important role in the evolution of all-electric technology. In relation to the proposed study, two areas of research and development would be worthwhile. The first is the test and evaluation of electric primary surface actuators. The second is in the area of secondary power control. As in the case of the advanced displays, the control of electric power for future all-electric aircraft will be dependent on ultra-reliable digital electronics. The all-electric flight control

system of the 1990's will use ring buses to efficiently distribute power to the electric motors and actuators on the aircraft. Remote-controlled, high-power solid state switches will control power to various parts of the aircraft. A total power failure cannot be tolerated; hence, FBW-quality digital control will again be a necessity.

APPENDIX A

EXAMPLES OF FINITE-STATE MACHINES FOR SPECIFYING FLIGHT CONTROL FUNCTIONS*

BACKGROUND

In many cases designers attempt to write software directly from an English language definition of the problem. Therefore, most of the design decisions and algorithm steps get buried in the software, the correctness of which is dependent solely on the intuition and ingenuity of the programmer. This poses two problems. First, most of the current program proving techniques cannot be applied because they require a formal mathematical specification of what the program is supposed to do. Second, if the algorithm has a design error, it is very difficult to detect.

We have proposed an approach that the algorithm be specified in terms of finite-state machine descriptions before writing the software so that the design decisions are made explicit and can be verified easily.

The digital advanced avionic system (DAAS) flight control program showed that describing mode logic as a finite-state machine was very effective in making design decisions visible and preventing errors of omission.

This appendix presents the details of three additional examples cited in section 3. The first example is an algorithm for selecting from three redundant sensor signals. It illustrates the use of a finite-state machine for exhibiting the structure of the algorithim. It also illustrates the dominance of one failure management mechanism over another.

The second example describes a three-channel synchronization method. It shows the need for failure effects analysis of the auxiliary hardware as it interfaces with the software.

---

*Portions of this research are supported by Honeywell IR&D programs.

The interaction between cross-channel voting and the testing of interchannel communications is studied in the third example. The study shows that three channels are adequate for detecting the first failure.

These specific examples were chosen to illustrate finite-state machine modeling; depending on final implementation of the ADFBW architecture, they may or may not be part of the recommended system.

SIGNAL SELECTION FOR THREE REDUNDANT SENSORS WITH VALIDITY FLAGS

This algorithm provides an example of the finite-state structure. The description is intended to be precise, complete, and clear to allow a design review and a proof of correctness by a walk-through demonstration.

A major part of this problem consists of combinational (nonsequential) logic. Mathematical (boolean) expressions of input variables have to be evaluated in order to determine the transitions of the finite-state machine. Similarly, mathematical expressions of input variables and the current state yield the output variables. In the three-sensor select problem, these mathematical expressions are of vital importance and strongly reflect the control engineering decisions. Therefore, it is essential that these expressions be explicitly derived and stated.

Based on the above reasoning, the recommended approach is to have a complete mathematical description of the solution, which would serve as a specification for the software to be written. This mathematical description consists of a finite-state machine description and some boolean algebra in the following example. In a control law problem, it may consist of arithmetic expressions denoting, for example, the transfer function.

The main advantage of a mathematical description is that it is a language easily understood by the control enginers. A systematically derived mathematical expression constitutes a proof in itself. It also highlights the control engineering decisions in the best possible manner.

Another advantage is that once the mathematical description is written, hardware/software tradeoffs and implementation allocations can be readily made.

Note that this approach is quite compatible with the current program verification techniques. The required input and output assertions can be readily obtained from the mathematical description. The use of such techniques may not be necessary, however, because as experience indicates, most errors are made in obtaining the correct description of what the software is supposed to do.

The entire process can be summarized as follows:

(1) Control engineering statement of problem. - An English language statement of the problem, specifying the input and output variables.

(2) Develop solution approach and informal algorithm(s) based on control engineering reasoning.

(3) Mathematical description. -

(a) Identify what information has to be preserved from one cycle to another. This constitutes the finite-state machine.

(b) Explicitly state which combinations of input variable cause a particular transition in the finite-state machine (this is based on control engineering decision). Complete the formal description of the finite-state machine.

(c) Each output variable is a function of input variables and the current state of the finite-state machine. Describe this function mathematically, taking into account all possible states of the finite-state machine and all possible combinations of the input variables.

(4) Hardware/software split. - Based on the results of the previous step, decide what portion is to be implemented in hardware and what in software.

(5) Design and verify the hardware.

(6) Design and verify the software. -

(a) Design software according to the specification in step 4.

(b) Derive input-output assertion to be used for formal program verification.

(c) Formally verify the program.

(Steps b and c may be omitted if not critical.)

Based on the above methodology, the remainder of this section presents a solution for the three-sensor select problem.

A SOLUTION OF THE THREE-SENSOR SELECT PROBLEM

Problem Statement

To produce an output signal from three sensor signals with validity flags sampled at each cycle.

Requirements and Approach

The three sensor signals and the three corresponding validity flags are directly wired to each computer. If a flag is invalid, assume that the sensor has failed. There may, however, be failure modes not detected by the validity mechanism. Hence, signal comparisions are also necessary. The differences of the signals are required to be within a fixed tolerance that is specific to the sensor.

The selection of the output depends only on the comparisons of the three signals. If the three signals are valid and compare within the tolerance, the median signal is chosen. If one pair of signals miscompare, the third signal is used but no fault is assigned. If two pairs of signals miscompare, then the signal common to the pairs is judged faulty and the average of the other two is taken as the selected signal. If there are three miscomparisons while all of the flags are valid or the situation is ambiguous, none of the data is used; the selected value from the previous cycle is chosen.

To avoid nuisance error indications a counting mechanism is used to determine failure when a fixed plurality of miscompares is exceeded. A sensor will be considered to be recovered if it compares favorably for the same plurality of cycles. The status of the validity flag and the count of miscompares determines the state of the sensor. This part of the algorithm is represented as a finite-state machine.

Since there are two mechanisms for monitoring failures, these must be shown to provide consistent determinations under all circumstances. It is conceivable, with time skew of the computer programs and momentary jitter in the flag signal, that the programs might disagree on the mode of failure, but this condition must not persist or allow differences in the selected signals.

Mathematical Description

The information to be preserved from one cycle to another is the state of the counting mechanism and the sensor status (i.e., failed or OK). A finite-state machine representation of this will look like figure A-1. The total number of states is 2 Max + 1. Starting from state 1, the machine moves right one state every time sensor A is considered "bad." A transition to the left occurs every time sensor A is considered "good." In case of indecision, there is no transition. These states can also be represented by the values of integer variable count-A in the range Max to 1. When the value of count-A reaches zero, sensor A is considered in a failed state. The sensor is not considered recovered until count A reaches Max. The counting up mechanism is exactly the reverse of the counting down.

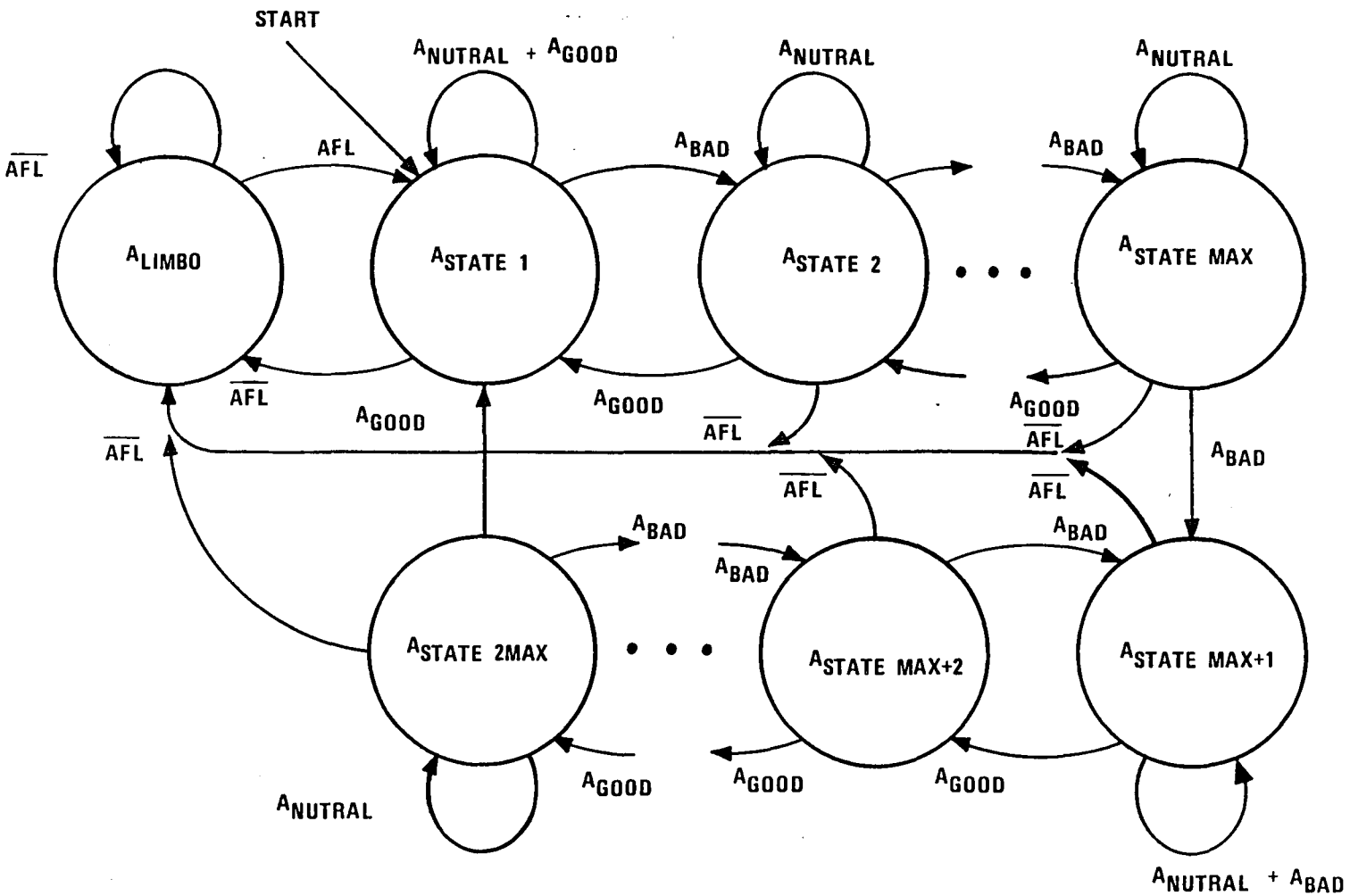
In addition to the above, there are other transitions based on the value of the validity flag only. All the states of this finite-state machine transition to state "limbo" whenever sensor A validity flag (Afl) is false. The only exit from limbo is to the start-up state (state 1) when Afl is "true." Notice that whenever Afl becomes "false," it essentially resets the finite-state machine. It is important to note that conditions Afl, Abad, Agood, and Anutral should be mutually exclusive. Based on this finite-state machine description, sensor A is called in a failed state if it is in any one of the following states: ALimbo, AstateMax+1, AstateMax+2, ..., Astate2Max. A boolean "Afail" is generated to denote this condition. It is true if the finite-state machine for sensor A is in any one of the above states. It is false otherwise. Booleans Bfail and Cfail are generated in a similar manner. Note that if Afl-false, it will always force Afail-true. There are similar, independent finite-state machines for sensor B and sensor C.

The objective is to derive boolean expressions for conditions Abad, Agood, and Anutral. These expressions should contain as variable only the inputs and the current (unadvanced) states of the finite-state machines for the other two sensors. First, let us list the inputs and define some intermediate variables:

a - real - value of sensor A

b - real - value of sensor B

c - real - value of sensor C



$$A_{FAIL} \triangleq (A_{LIMBO}) + (A_{STATE\ MAX+1}) + (A_{STATE\ MAX+2}) + \ldots + (A_{STATE\ 2\ MAX})$$

Figure A-1. - Finite-state machine for sensor A.

Afl - boolean - validity flag of sensor A

Bfl - boolean - validity flag of sensor B

Cfl - boolean - validity flag of sensor C

AB boolean - True: ABS(a-b) < Tolerance

AC boolean - True: ABS(a-c) < Tolerance

BC boolean - True: ABS(b-c) < Tolerance

A truth table is given in table A-1. The expressions for Abad, Agood, and Anutral are directly obtained from this table.

Abad: Boolean; should be true when Afl-true and there is sufficient reason to believe that sensor A is bad.

$$Abad = Afl \{\overline{Bfl} \cdot Cfl \cdot \overline{AC} \cdot \overline{Cfailp} + Bfl \cdot \overline{Cfl} \cdot \overline{AB} \cdot \overline{Bfailp} + Bfl \cdot Cfl \cdot \overline{AB} \cdot AC \cdot (BC + \overline{BC} \cdot \overline{Bfailp} \cdot \overline{Cfailp})\}$$

Agood: Boolean; true whenever there is sufficient reason to believe that sensor A is good and Afl-true.

$$Agood = Afl \{\overline{Bfl} \cdot \overline{Cfl} + \overline{Bfl} \cdot AC + Cfl \cdot AB + Bfl \cdot Cfl \cdot (AB \cdot \overline{BC} + AC \cdot \overline{BC} + AB \cdot BC \cdot AC)\}$$

Anutral: Boolean; true whenever there is insufficient reason for either Agood-true or Abad-true.

$$Anutral = Afl \cdot \overline{Agood} \cdot \overline{Bgood}$$

Afailp: Value of Afail at the end of previous cycle.

Bfailp: Value of Bfail at the end of previous cycle.

Similar expressions can be obtained for Bgood, Bbad, and Bnutral; and Cgood, Cbad, and Cnutral. This completes the finite-state machine descriptions. Note that if the three finite-state machines are advanced simultaneously, there is no need to define Afailp, Bfailp, and Cfailp. This is standard practice in hardware implementations of finite-state machines.

There are two outputs:

Sensor-type-fails. - Boolean; true when there is sufficient reason to believe that all the three sensors have failed, or it cannot be known which sensor is good.

sensor-type-fail - Afail Bfail Cfail

TABLE A-1. - TRUTH TABLE FOR AGOOD, ABAD, AND ANUTRAL

Inputs: Afl, Bfl, Cfl, AB, BC, AC, Bfailp, and Cfailp

I. Afl-true

Bfl	Cfl	AB	BC	AC	Agood	Abad	Anutral
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	0	0
0	0	1	0	1	1	0	0
0	0	1	1	0	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	$\overline{\text{Cfailp}}$	Cfailp
0	1	0	0	1	1	0	0
0	1	0	1	0	0	$\overline{\text{Cfailp}}$	Cfail
0	1	0	1	1	1	0	0
0	1	1	0	0	0	$\overline{\text{Cfailp}}$	Cfail
0	1	1	0	1	1	0	0
0	1	1	1	0	0	$\overline{\text{Cfailp}}$	Cfail
0	1	1	1	1	1	0	0

Cfailp $\triangleq$ value of Cfail at end of previous cycle

Bfailp $\triangleq$ value of Bfail at end of previous cycle

TABLE A-1. - Concluded

Bfl	Cfl	AB	BC	AC	Agood	Abad	Anutral
1	0	0	0	0	0	$\overline{\text{Bfailp}}$	Bfailp
1	0	0	0	1	0	$\overline{\text{Bfailp}}$	Bfailp
1	0	0	1	0	0	$\overline{\text{Bfailp}}$	Bfailp
1	0	0	1	1	0	$\overline{\text{Bfailp}}$	Bfailp
1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	0	1	0	0
1	0	1	1	1	1	0	0
1	1	0	0	0	0	$\overline{\text{N}}$	N
1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	0	0
1	1	1	0	1	1	0	0
1	1	1	1	0	0	0	1
1	1	1	1	1	1	0	0

N = Bfail • Cfail

II. Afl-false

Agood = Abad = Anutral = false
regardless of other inputs

Selected-value. - Real; denotes the selected signal value for this cycle. The following variables are used as inputs:

Afail, Bfail, Cfail, AB, BC, AC, a, b, c, past-value

Table A-2 gives the selected-value for all possible combinations of the input variables. It can be easily verified that the eight boolean conditions are indeed mutually exclusive and that they account for all possible (64) combinations of the six input booleans.

Hardware-Software Split

There are various options available here, as listed below:

(a) The entire algorithm can be implemented in hardware. This will be a very straightforward but tedious design.

(b) Part of the algorithm can be implemented in hardware, such as evaluating the booleans AB, BC, AC, or the finite-state machines.

(c) The entire algorithm can be implemented in software.

For the purpose of an example, the third approach is used here. The algorithm steps are:

(1) Evaluate any intermediate variables.

(2) Process data for sensor A (i.e., advance finite-state machine).

(3) Process data for sensor B.

(4) Process data for sensor C.

(5) Evaluate outputs.

The next step in developing software for this function would be to prepare the HIPO charts from the finite-state machine description.

Verification

A walk-through of the algorithm can be used to show that, as a single computer program, it is technically correct. The compare and select functions are not complicated so their implementations can be tested for all combinations of boolean inputs and a reasonable representation of the combinations of real inputs.

TABLE A-2. - GENERATING OUTPUT VARIABLES.

Condition	Selected-Value
1. $\overline{Afail} \cdot \overline{Bfail} \cdot \overline{Cfail} \cdot AB \cdot BC \cdot AC = true$	median (a, b, c)
2. $\overline{Bfail} \cdot \overline{Cfail} \cdot (Afail \cdot BC + \overline{Afail} \cdot \overline{AB} \cdot BC \cdot \overline{AC}) = true$	$\frac{b + c}{2}$
3. $\overline{Afail} \cdot \overline{Cfail} \cdot (Bfail \cdot AC + \overline{Bfail} \cdot \overline{AB} \cdot \overline{BC} \cdot AC) = true$	$\frac{a + c}{2}$
4. $\overline{Afail} \cdot \overline{Bfail} \cdot (Cfail \cdot AB + \overline{Cfail} \cdot AB \cdot \overline{BC} \cdot \overline{AC}) = true$	$\frac{a + b}{2}$
5. $\overline{Afail} \cdot (\overline{Bfail} \cdot Cfail + \overline{Bfail} \cdot \overline{Cfail} \cdot AB \cdot \overline{BC} \cdot AC) = true$	a
6. $\overline{Bfail} \cdot (Afail \cdot Cfail + \overline{Afail} \cdot \overline{Cfail} \cdot AB \cdot BC \cdot \overline{AC}) = true$	b
7. $\overline{Cfail} \cdot (Afail \cdot Bfail + \overline{Afail} \cdot \overline{Bfail} \cdot \overline{AB} \cdot BC \cdot AC) = true$	c
8. $Afail \cdot Bfail \cdot Cfail + \overline{Afail} \cdot \overline{Bfail} \cdot \overline{Cfail} \cdot \overline{AB} \cdot \overline{BC} \cdot \overline{AC} +$ $Afail \cdot \overline{Bfail} \cdot \overline{Cfail} \cdot \overline{BC} + Afail \cdot Bfail \cdot Cfail \cdot AC +$ $\overline{Afail} \cdot \overline{Bfail} \cdot Cfail \cdot \overline{AB}$	past-value

ANALYSIS OF A THREE-CHANNEL SYNCHRONIZATION MECHANISM

One approach to frame synchronization is to use dedicated hardware, external to the computers, to provide signals to simultaneously release the computers from the halt instruction in each copy of the software. These schemes require very careful failure modes and effects analysis to show that no single failure in this external hardware or in the computers results in total system failure.

The Configuration

The configuration for synchronization is shown in figure A-2. Each block of hardware communicates with the two other blocks and its respective computer. In addition, there is a flip-flop which is set by the local computer program when its execution leaves the initialization phase. This flip-flop may be read by the other computers. The corresponding boolean variables are called the right_up_and_ready, local_up_and_ready, and left_up and_ready to distinguish them from the ready signals available from the hardware logic shown in figure A-3.

After the hardware is reset, the real-time counter counts for 25 msec and sets the ready flip-flop. If the counting continues through the overcount period, the overcount flip-flop is set. The hardware produces the halt release signal from the following two terms, which are combined at the final or-gate. For two or three computers,

```
halt_release = ((left_overcount AND local-overcount) OR right_ready)
     AND ((right_overcount AND local_overcount) OR left_ready)
     AND local_ready
```

But to provide for the case in which two computers fail, we need the term

```
halt_release = ((NOT left_overcount OR
          NOT local_overcount) AND NOT right_ready)
     AND ((NOT right_overcount OR
          NOT Local_overcount) AND NOT Left_ready)
     AND local_overcount
```

The hardware is implemented so that the power-down or broken wire case appears as ready = true and overcount = true to the other channels.

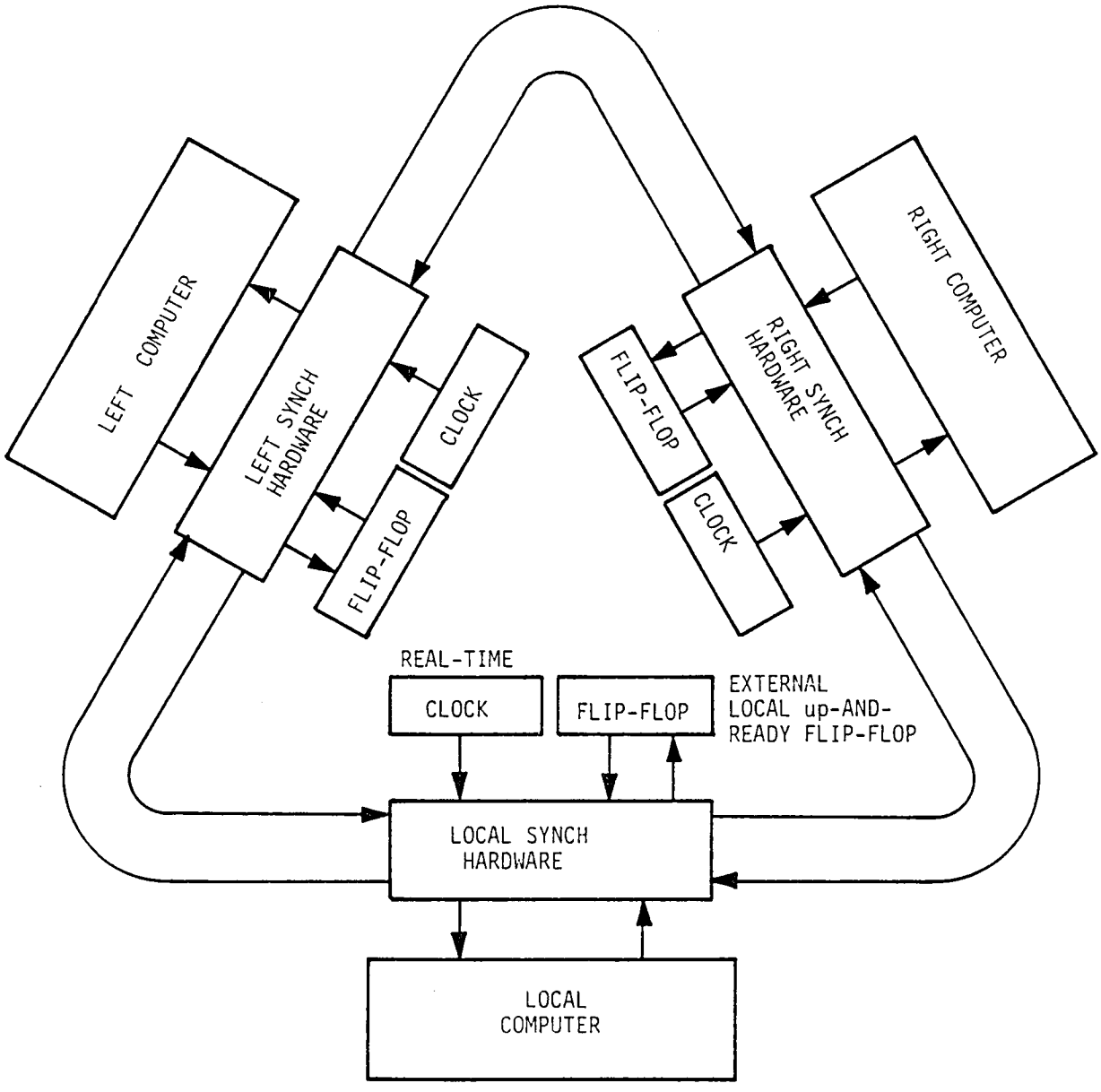


Figure A-2. - Configuration for synchronization.

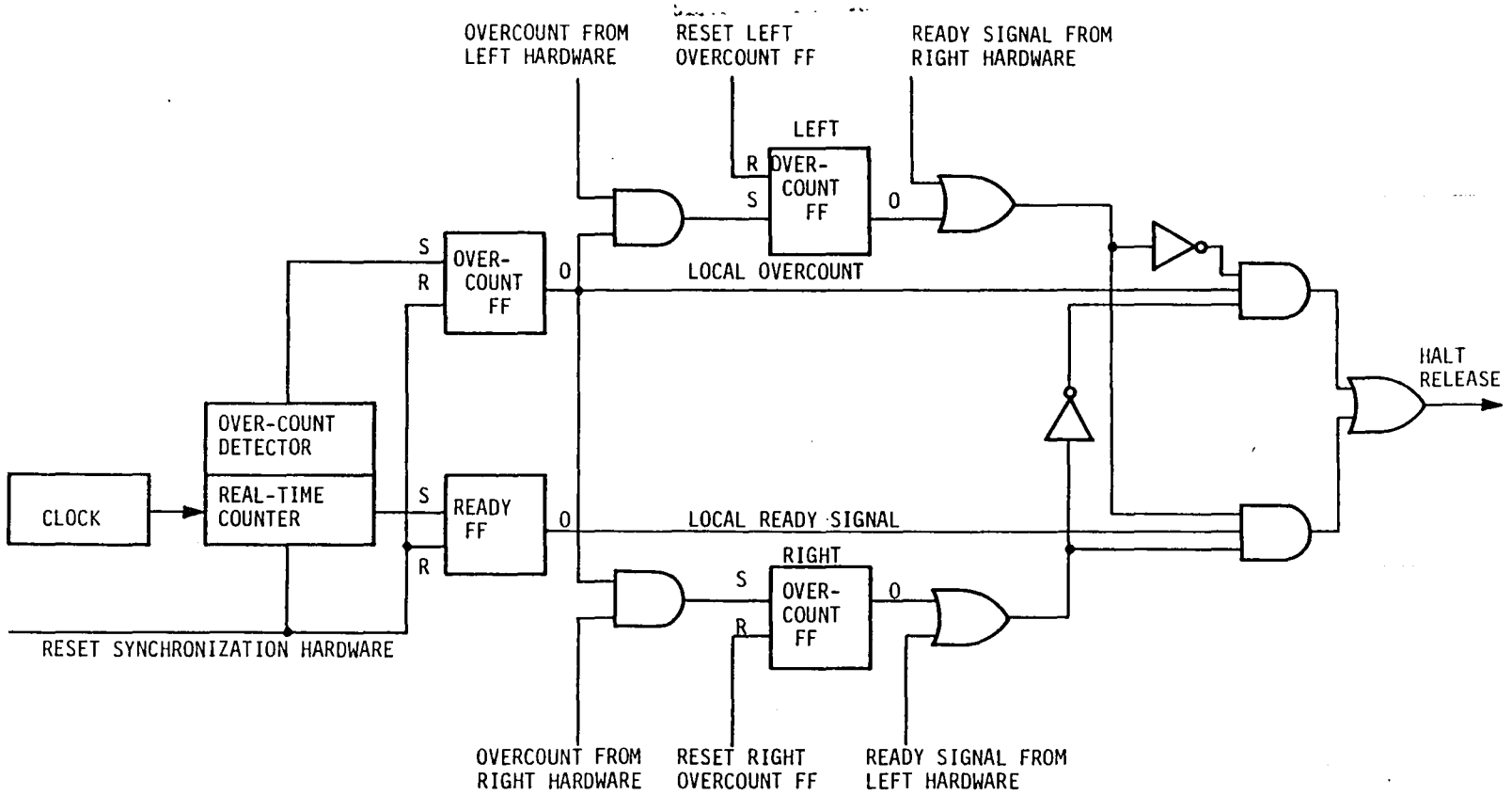


Figure A-3. - Hardware logic for synchronization.

Initial Synchronization

The process of initially synchronizing with a computer that is already running in the frame is illustrated in figure A-4. If the right or left computers are already running, they have passed the instruction in the initial leg of the program that sets the up_and_ready flip-flop. The local computer detects that this flip-flop is set and waits in the starting leg until the running computer passes the halt and resets the ready flip-flop in the synchronizing hardware. This is detected in the wait loop and the local computer is released. The time required for the program to run from the ready reset to the beginning of the loop is balanced to maintain the synchronization.

Analysis of Frame Synchronization

The auxiliary hardware has three states that are defined by the ready and ovecount flip-flops. These are:

(1) NOT ready AND NOT overcount (the program is in the 25 μsec main program loop)

(2) Ready AND NOT overcount (the clock has timed past 25 μsec; normally the program is at halt, waiting for the other ready's.

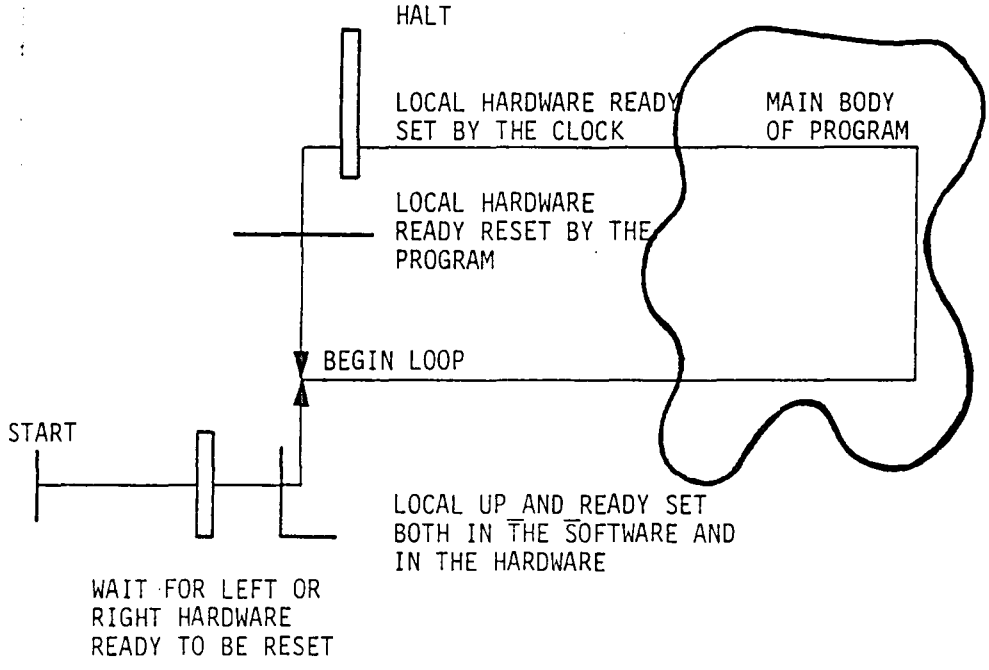


Figure A-4. - Initial synchronization.

(3) Ready AND overcount (the clock has counted past the overcount period and has set the overcount flip-flop.)

The parallel operation of the computer and the auxiliary hardware is illustrated by the Petri net diagram in figure A-5. It represents the operation of the local computer with the right computer; the left computer is not turned on. Places P_1 to P_3 with transitions T_1 to T_3 represent the operation of the local clock. Places P_4 and P_5 with transitions T_3 and T_4 are the states of overcount flip-flop. Places P_6 and P_7 with transitions T_5 and T_6 are the states of the ready flip-flop. Transition T_6 represents the halt-release logic of

halt-release = local-ready AND (right-ready OR
local-overcount)

The local computer is at halt in place P_{10} but is running and crosses the reset command in the software at transition T_{11}.

Three events govern the operation of the computer:

(1) Halt release is issued, ready and overcount flip-flops are reset, clock is reset

(2) Clock sets ready flip-flop

(3) Clock sets overcount flip-flop

Thus, there are trivial relations between the events and the states. These are shown in table A-3 and represent the 27 states of the three computers. This table merely confirms the consistency of the operation and defines the states in which a release command is output. If only computers A and B are operating, C reports that it is in the state ready AND overcount. This limits the operation of the system to the states and transitions shown in table A-4. With only one computer operating, there are only the three remaining states, illustrated in table A-5.

The release-enable output listed in the last column of the tables may be represented by

release a enable = [a(1) AND NOT [b(0) OR
c(0)]] OR [a(2) AND
NOT [(b(0) AND c(1)) OR
(b(1) AND c(0))]]

This can be verified to be equivalent to the boolean logic of the hardware listed previously.

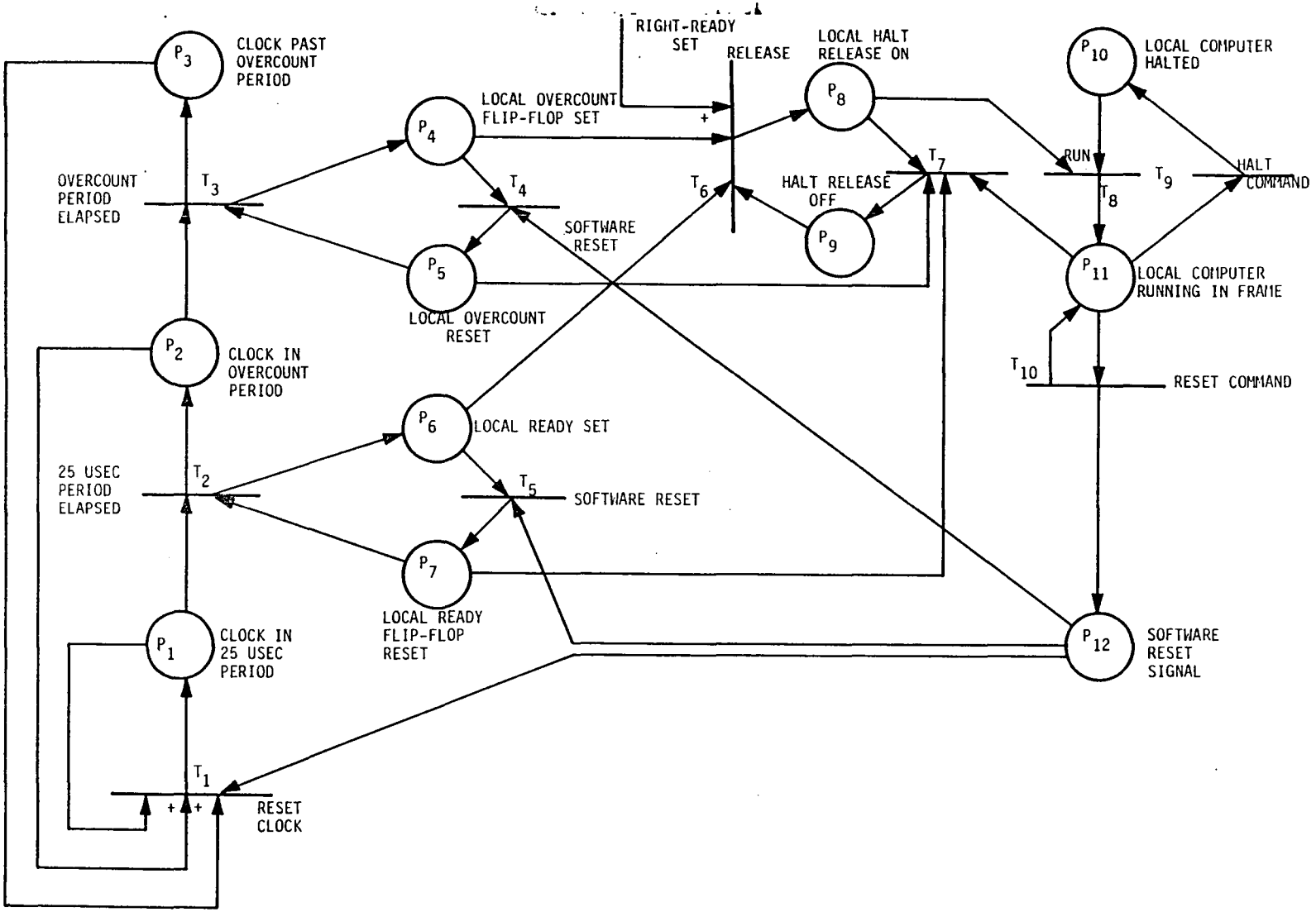


Figure A-5. - Petri net for local computer.

TABLE A-3. - ALL THREE COMPUTER GUIDES

State \ Event	NOT Ready NOT Overcount	1. Ready	2. Ready Overcount	A Released	A Ready	A Overcount	B Released	B Ready	B Overcount	C Released	C Ready	C Overcount	Release Enabled
0	A B C			-	5	-	-	4	-	-	3	-	
1		A B C		12	-	17	13	-	16	14	-	15	A B C
2			A B C	9	-	-	10	-	-	11	-	-	A B C
3	A B	C		-	13	-	-	12	-	0	-	6	
4	A C	B		-	14	-	0	-	7	-	12	-	
5	B C	A		0	-	8	-	14	-	-	13	-	
6	A B		C	-	23	-	-	21	-	0	-	-	C
7	A C		B	-	25	-	0	-	-	-	22	-	B
8	B C		A	0	-	-	-	26	-	-	24	-	A
9	A		B C	-	18	-	6	-	-	7	-	-	B C
10	B		A C	6	-	-	-	19	-	8	-	-	A C
11	C		A B	7	-	-	8	-	-	-	20	-	A B
12	A	B C		-	1	-	3	-	22	4	-	21	
13	B	A C		3	-	24	-	1	-	13	-	23	
14	C	A B		4	-	26	5	-	25	-	1	-	
15		A B	C	21	-	19	23	-	18	14	-	-	A B C
16		A C	B	22	-	20	13	-	-	25	-	18	A B C
17		B C	A	12	-	-	24	-	20	26	-	19	A B C
18		A	B C	9	-	2	23	-	-	25	-	-	A B C
19		B	A C	21	-	-	10	-	2	26	-	-	A B C
20		C	A B	22	-	-	24	-	-	11	-	2	A B C
21	A	B	C	-	15	-	6	-	9	4	-	-	
22	A	C	B	-	16	-	3	-	-	7	-	9	
23	B	A	C	6	-	10	-	15	-	5	-	-	
24	B	C	A	3	-	-	-	17	-	8	-	10	
25	C	A	B	7	-	11	5	-	-	-	16	-	
26	C	B	A	4	-	-	8	-	11	-	17	-	

TABLE A-4. - COMPUTERS A AND B OPERATING

State \ Event	NOT Ready NOT Overcount	1. Ready	2. Ready Overcount	A Released	A Ready	A Overcount	B Released	B Ready	B Overcount	C Released	C Ready	C Overcount	Release Enabled
0													
1													
2			A B C	9	-	-	10	-	-	11	-	-	A B C
3													
4													
5													
6	A B		C	-	23	-	-	21	-	0	-	-	C
7													
8													
9	A		B C	-	18	-	6	-	-	7	-	-	B C
10	B		A C	6	-	-	-	19	-	8	-	-	A C
11													
12													
13													
14													
15		A B	C	21	-	19	23	-	18	14	-	-	A B C
16													
17													
18		A	B C	9	-	2	23	-	-	25	-	-	A B C
19		B	A C	21	-	-	10	-	2	26	-	-	A B C
20													
21	A	B	C	-	15	-	6	-	9	4	-	-	
22													
23	B	A	C	6	-	10	-	15	-	5	-	-	
24													
25													
26													

TABLE A-5. - COMPUTER A OPERATING

State \ Event	NOT Ready NOT Overcount	1. Ready	2. Ready Overcount	A Released	A Ready	A Overcount	B Released	B Ready	B Overcount	C Released	C Ready	C Overcount	Release Enabled
0													
1													
2			A B C	9	-	-	10	-	-	11	-	-	A B C
3													
4													
5													
6													
7													
8													
9	A		B C	-	18	-	6	-	-	7	-	-	B C
10													
11													
12													
13													
14													
15													
16													
17													
18		A	B C	9	-	2	23	-	-	25	-	-	A B C
19													
20													
21													
22													
23													
24													
25													
26													

The Results

By this analysis, we have verified the consistency of the synchronization scheme for all states and all events and have verified the release logic equations. The next step is to study the failure effects of the auxiliary hardware to show that there are no single failures which will cause a persistent unsynchronized condition or will cause the system to fail by some other response. These results are not reported here.

CROSS-CHANNEL VOTING AND TESTING OF INTERCHANNEL COMMUNICATIONS

This study describes the interchannel communication typical of a frame-synchronized triplex system (ref. A1). The configuration is shown in figure A-6. Each computer communicates to the others through a single transmitter, which sends the same signals to receivers at each of the other computers. Thus, the sending computer-transmitter cannot originate two different signals. Asymmetry in the communications can be caused only by errors in the receivers or the receiving computer. This approach eliminates the concerns raised in reference A2, which is carried needlessly into SIFT (ref. A3 to A6). According to reference 2, we need four computers to detect one error if the originating computer sends differing signals to the others. This is not the case for the configuration shown in figure A-6.

The Analysis

The approach is by brute force. Assume that any one of the 12 boxes in figure A-6 produces errors and then follow these errors through two levels of data exchange. Only one unit is assumed faulty. Errors are detected by a sum check on the data transmissions and by comparisons of computer outputs from some active computation. The error syndromes after the initial data exchange are listed in table A-6; the final syndromes resulting from the exchange of the initial observations are shown in table A-7. After the first exchange the syndromes allow a computer to detect errors in the foreign computers or the communications channels, but cannot distinguish between errors in the computers, transmitters, or receivers. After the second round of data

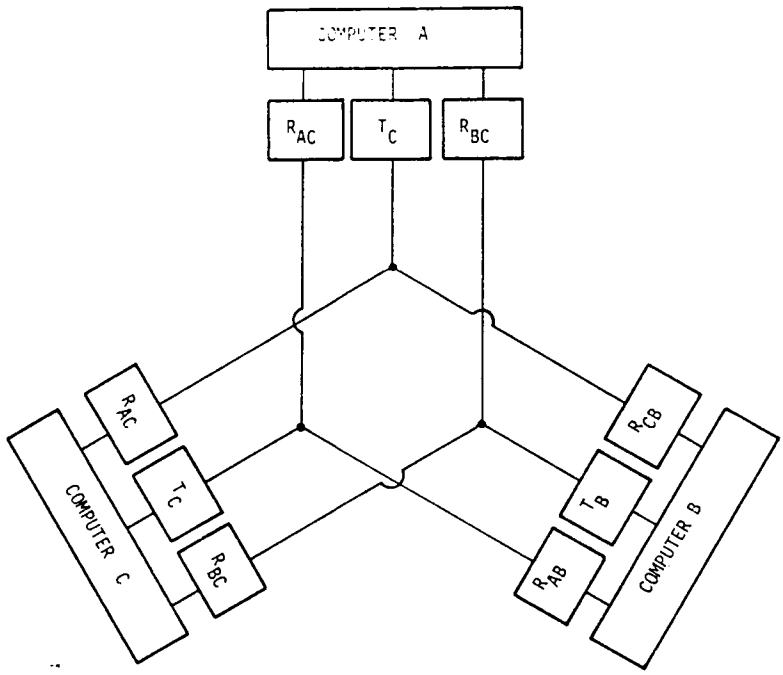


Figure A-6. - Communication among synchronized channels.

interchange, the syndromes distinguish receiver errors and computer-transmitter errors; the local computer, if okay, can determine that its transmitter is causing errors.

In the second round of communications, a computer will receive a word that indicates an error in the left or right path, or its own transmitter. The transmission over an erroneous path is indicated by an X in table A-7.

The algorithm is summarized in table A-8. There is a jump in the frame of reference from the initial observation to the final analysis in table A-6; if the right channel decides that its left channel is in error, then the local channel will interpret this decision to mean that it is in error.

TABLE A-6. - INITIAL FAULT OBSERVATIONS

EVENT - ERROR IN / OBSERVATION	SUM CHECK ON C TO A	SUM CHECK ON B TO A	COMPARISON CB AT A	COMPARISON AC AT A	COMPARISON BA AT A	CONCLUSION OF COMPUTER A	SUM CHECK ON A TO B	SUM CHECK ON C TO B	COMPARISON AC AT B	COMPARISON BA AT B	COMPARISON CB AT B	CONCLUSION OF COMPUTER B	SUM CHECK ON B TO C	SUM CHECK ON A TO C	COMPARISON BA AT C	COMPARISON CB AT C	COMPARISON AC AT C	
1 C_A	X	X	X	X	X	PROBABLY SOMETHING WRONG	X	OK	FAIL	FAIL	OK	ERROR IN A TO B IN COMPUTER A	OK	X	FAIL	OK	FAIL	ERROR IN A TO C OR IN COMPUTE A
2 T_A	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN A TO B OR IN COMPUTER A	OK	FAIL	X	OK	X	ERROR IN A TO C OR IN COMPUTER A
3 R_{AB}	OK	FAIL	X	OK	X	ERROR IN B TO A OR IN COMPUTER B	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM
4 R_{CA}	FAIL	OK	X	X	OK	ERROR IN C TO A OR COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM
5 C_B	OK	X	FAIL	OK	FAIL	ERROR IN B TO A OR COMPUTER B	X	X	X	X	X	PROBABLY SOMETHING WRONG	X	OK	FAIL	FAIL	OK	ERROR IN B TO C OR IN COMPUTER B
6 T_B	OK	FAIL	X	OK	X	ERROR IN B TO A OR COMPUTER B	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN B TO C OR IN COMPUTER B
7 R_{CB}	OK	OK	OK	OK	OK	NO PROBLEM	OK	FAIL	X	OK	X	ERROR IN C TO B OR IN COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM
8 R_{AB}	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN A TO B OR IN COMPUTER A	OK	OK	OK	OK	OK	NO PROBLEM
9 C_C	X	OK	FAIL	FAIL	OK	ERROR IN C TO A OR IN COMPUTER C	OK	X	FAIL	OK	FAIL	ERROR IN C TO B OR IN COMPUTER C	X	X	X	X	X	PROBABLY SOME - THING WRONG
10 T_C	FAIL	OK	X	X	OK	ERROR IN C TO A OR IN COMPUTER C	OK	FAIL	X	OK	X	ERROR IN C TO B OR IN COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM
11 R_{AC}	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM	OK	FAIL	X	OK	X	ERROR IN A TO C OR IN COMPUTER A
12 R_{BC}	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN B TO C OR IN COMPUTER B

TABLE A-7. - SECOND DATA INTERCHANGE

EVENT / REPORT	OBSERVATION OF A	REPORT OF B TO A	REPORT OF C TO A	CONCLUSION OF COMPUTER A	REPORT OF A TO B	OBSERVATION OF B	REPORT OF C TO B	CONCLUSION OF COMPUTER B	REPORT OF A TO C	REPORT OF B TO C	OBSERVATION OF C	CONCLUSION OF COMPUTER C
1 C_A	X	X	X	X	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A
2 T_A	OK	A TO B OR C_A	A TO C OR C_A	ERROR IN T_A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A
3 R_{BA}	B TO A OR C_B	X	OK	ERROR IN R_{BA}	B TO A OR C_B	OK	OK	ERROR IN R_{BA}	B TO A OR C_B	OK	OK	ERROR IN R_{BA}
4 R_{CA}	C TO A OR C_C	OK	X	ERROR IN R_{AC}	C TO A OR	OK	OK	ERROR IN R_{AC}	C TO A OR C_C	OK	OK	ERROR IN R_{AC}
5 C_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B	X	X	X	X	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B
6 T_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B	B TO A OR C_B	OK	B TO C OR C_B	ERROR IN T_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B
7 R_{CB}	OK	C TO B OR C_C	OK	ERROR IN R_{CB}	OK	C TO B OR C_C	X	ERROR IN R_{CB}	OK	C TO B OR C_C	OK	ERROR IN R_{CB}
8 R_{AB}	OK	A TO B OR C_A	OK	ERROR IN R_{AB}	X	A TO B OR C_A	OK	ERROR IN R_{AB}	OK	A TO B OR C_A	OK	ERROR IN R_{AB}
9 C_C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	X	X	X	X
10 T_C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	C TO B OR C_C	PI	ERROR IN T_C
11 R_{AC}	OK	OK	A TO C OR C_A	ERROR IN R_{AC}	OK	OK	A TO C OR C_A	ERROR IN R_{AC}	X	OK	A TO C OR C_A	ERROR IN R_{AC}
12 R_{BC}	OK	OK	B TO C OR C_B	ERROR IN R_{BC}	OK	OK	B TO C OR C_B	ERROR IN R_{BC}	OK	X	B TO C OR C_B	ERROR IN R_{BC}

TABLE A-8. - SUMMARY OF FAILURE ANALYSIS ALGORITHM

SUN CHECK RIGHT TO LOCAL	SUN CHECK LEFT TO LOCAL	COMPARISON OF RIGHT AND LEFT	COMPARISON OF LOCAL AND RIGHT	COMPARISON OF LEFT AND LOCAL	INITIAL FAULT OBSERVATION	REPORT FROM THE RIGHT	OBSERVATION OF LOCAL	REPORT FROM THE LEFT	FAULT DIAGNOSIS
X	OK	FAIL	FAIL	OK	RIGHT CHANNEL	OK	OK	OK	OK
FAIL	OK	X	X	OK	RIGHT CHANNEL	X	RIGHT	RIGHT	RIGHT CHANNEL
OK	FAIL	X	OK	X	LEFT CHANNEL	LEFT	LEFT	X	LEFT CHANNEL
OK	X	FAIL	OK	FAIL	LEFT CHANNEL	LOCAL	OK	LOCAL	LOCAL TRANSMITTER
OK	OK	OK	OK	OK	OK	LOCAL	OK	OK	LOCAL TO RIGHT RECEIVER
ALL OTHERS		→		→	LOCAL CHANNEL	LEFT	OK	OK	RIGHT TO LEFT RECEIVER
						OK	OK	RIGHT	LEFT TO RIGHT RECEIVER
						OK	LEFT	X	LEFT TO LOCAL RECEIVER
						X	RIGHT	OK	RIGHT TO LOCAL RECEIVER
						ALL OTHERS	→		LOCAL CHANNEL

(X = OK OR FAIL)

INITIAL FAULT OBSERVATION

(X = OK OR LEFT OR RIGHT OR LOCAL OR ERROR)

SECOND DATA INTERCHANGE

APPENDIX A REFERENCES

A1. Bender, M.A.; to W.A. Becker: DAFICS Software DS-II-Redundant Signal Monitoring and Management Portion. Internal Honeywell Avionics Division memo, February 10, 1981.

A2. Pease, M.; Shostak, R.; and Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM, vol. 27, no. 2, April 1980, pp. 228-234.

A3. Weinstock, C.B.: SIFT: System Design and Implementation. 10th Int. Symp. on Fault-Tolerant Computing, October 1980.

A4. Goldberg, J.: SIFT: A Provable Fault-Tolerant Computer for Aircraft Flight Control. Information Processing 80, (IFIP 1980), S. H. Lavington, ed., pp. 151-156.

A5. Melliar-Smith, P.M.; and Schwartz, R.L.: Current Progress on the Proof of SIFT. 11th Annual Int. Symp. on Fault-Tolerant Computing, Portland, Maine, June 1981.

A6. Melliar-Smith, P.M.; and Schwartz, R.L.: Hierarchical Specification on the SIFT Fault Tolerant Flight Control System. SRI-International manuscript.

A7. Levitt, K.N.: Software Validation and Verification Techniques. AGARD Lecture Series no. 109, pp. 5-1 to 5-9.

APPENDIX B

SAMPLE IRON BIRD TEST PLAN

INTRODUCTION

A sample test plan to validate the advanced digital fly-by-wire (ADFBW) in the S-3A iron bird is given in this appendix. The sample plan is used to illustrate the typical test procedures and to identify those tasks for which automation is essential for validating flight-critical digital systems and beneficial in reducing test time and cost. To illustrate the potential test time savings of the automated iron bird, test times are estimated for both the manual approach and the automated approach based on the same number of test cases to be conducted.

TEST DESCRIPTION AND TEST TIME ESTIMATION

The iron bird testing will be conducted in two phases. The two phases are defined as follows:

(1) Phase A. Phase A will test the ADFBW system's hardware and software open-loop performance. Aircraft dynamics will not be included. The purposes of this test phase are to:

- o Demonstrate compatibility among ADFBW systems and with aircraft interfacing systems
- o Verify static gains between stick/pedal and control surfaces
- o Verify system software

(2) Phase B. The tests conducted during this phase will evaluate the ADFBW closed-loop performance. Closed-loop testing will be accomplished with simulated aircraft dynamics. Testing with pilot-in-the-loop is possible by driving flight instruments with simulated aircraft response variables. The purposes of this test phase are to:

(1) Validate the analytically predicted stability of the augmentation mode

(2) Validate the fault-tolerance performance of the system

(3) Eliminate infant-mortality type failures by an accelerated life test procedure

(4) Evaluate handling qualities for normal and degraded modes operation by pilots

Phase A--Open-Loop Tests

Three tasks are identified in this phase: system interface test, static gain test, and software validation.

System interface test. - This test will be performed to verify that all subsystems are interfaced properly. The test procedure will include setting electrical and hydraulic power supplies at various loading levels and permissible limits to verify that the ADFBW system's performance will not result in undesirable or unsatisfactory operation. Because of the high degree of manual interpretation on the operational status of the system and the small number of test cases involved, a manual approach to carry out the test procedures will be adequate. The total test time to complete this task is estimated to be 80 hours.

Static gain test. - The static gain tests are performed to verify and evaluate the following:

(1) End-to-end gains--the degree of surface output per pound of control stick/pedal input

(2) Nonlinear effects such as hysteresis, deadband, and saturation

(3) Gain scheduling

All control axes will be evaluated. The procedure will include applying step inputs at control stick-pedal to evaluate end-to-end gains, applying function generator at different amplitudes to evaluate nonlinear effects, and setting airspeed at different levels for evaluation of gain scheduling.

A total of 100 test cases are estimated. The test times are estimated at 50 hours for manual operation and 16 hours for automated operation.

Software validation. - Redundancy management and control modes switching logic functions which are modeled by the finite-state machine can be validated in the iron bird by inserting hardware events into the software

structure to verify that all state transitions and outputs of all states are correct. For software that performs data transformation functions (e.g., control law and filter computation) which are not modeled by the finite-state machine, the software can be validated by frequency response and open-loop static gain tests in the iron bird. Since the volume of test cases for this task is quite high, automation is essential. The number of test cases and test times are estimated as summarized in table B-1.

Phase B--Closed-Loop Tests

Four tasks are identified in this phase: stability test, fault tolerance test, accelerated life test, and pilot-in-the-loop test.

Stability tests. - The stability performance of the augmentation mode as predicted by analyses will be validated in this test. Actual hardware such as sensors, electronics, and actuators will be included on the iron bird to eliminate error included in the analytical predictions owing to nonlinearities and other math modeling problems associated with these components. The typical test procedures will include the following:

TABLE B-1. - SOFTWARE VALIDATION TEST TIMES ESTIMATION

Software Function	Estimated No. of Test Cases	Estimated Test Time		Method
		Automated	Manual	
Control mode logic	500	4 hours	125 hours	Finite-state machine
Redundancy management	2000	16 hours	500 hours	Finite-state machine
Control laws	300	8 hours	15 hours	Frequency response and static gain
Total		28 hours	640 hours	

(1) Time response. - Apply steps and doublets into the control stick/pedal to observe short period and phugoid modes of the closed-loop system

(2) Frequency response. - Evaluate phase and gain margins by applying a sine wave into the open-loop system

(3) Closed-loop system damping. - Apply a sine wave at a frequency equal to the closed-loop phugoid and short period frequencies to observe system damping.

The number of test cases is based on the combination of flight conditions and number of frequency points, which are estimated to be on the order of 300 cases. Total test times estimated for this task are 60 hours for automated testing and 120 hours for manual testing.

Fault tolerance test. - The fault tolerance test is conducted by inserting multiple hardware faults into the system. Test cases will be designed based on the fault-tree of the system. The purpose of this test is to verify the fault-tree topology of the system which is used to predict the system's reliability. The ADFBW system's fault detection, reconfiguration, and annunciation features will be demonstrated. The ability of the system to operate under the fault conditions with no adverse transients will also be demonstrated.

Because of the complexity of the fault-tree structure and the importance of this test to validate the ultra-reliability requirement, a large number of fault combinations will be evaluated. A total of 20 000 fault combinations is estimated. Assuming 30 seconds per test case using automation, the total test time will be on the order of 170 hours. If automation is not available, the total test time will be on the order of 5000 hours.

Accelerated life test. - The purposes of the accelerated life test are to:

(1) Validate system performance

(2) Perform system fatigue tests

(3) Monitor component reliability

(4) Identify and eliminate design and implementation errors which are major contributors of system unreliability

All maneuver profiles, together with combinations of aircraft conditions and environments, will be evaluated. To compress the total test time, extreme environmental stressors will be applied to various subsystems to induce actual failures. All test cases will be repeated and cycled to induce fatigue type failures. A high degree of automation, such as the use of a robotic system to operate the aircraft in the iron bird, is required for this task.

A total of 200 test cases is estimated. These test cases will be repeated and cycled until a high degree of confidence is obtained that the system is free of design and implementation errors. The total test time for this task is estimated to be on the order of 1000 hours.

Pilot-in-the-loop test. - Normal and failure modes operation of the system will be used to demonstrate the handling qualities of airplanes with the ADFBW system and the augmentation mode engaged. The output data from each test will include pilot ratings and comments on the workload required to obtain satisfactory aircraft performance for the normal and degraded modes. The clarity and adequacy of fault annunciation will also be evaluated.

The total test time estimated for this task is 160 hours. No automation is required for this task.

SUMMARY

The total iron bird test times to validate the ADFBW system are summarized in table B-2. It can be shown that automation will offer substantial savings in test times. Using the recommended automated test plan, the total test time to validate the ADFBW system is estimated to be 1548 hours. The equivalent test time to perform the total test program manually is estimated to be 7100 hours.

TABLE B-2. - TEST TIMES SUMMARY

Test Task	Manual	Automated	Estimated Test Time (hours)	
			Manual	Automated
Phase A - Open loop				
o System interface	X		80	80
o Static gain		X	100	50
o Software validation		X	640	28
Phase B - Closed loop				
o Stability test		X	120	60
o Fault tolerance test		X	5000	170
o Accelerated life test		X	1000	1000
o Pilot-in-the-loop test	X		160	160
			7100	1548

REFERENCES

1. Integrated Application of Active Controls (IAAC) Technology to an Advanced Subsonic Transport Project--Current and Advanced ACT Control System Definition Study. Prepared under Contract NAS1-15325, NASA Contractor Report 165631, December 1980.

2. Luedde, W.J.: The Use of Separated Multi Function Inertial Sensors for Flight Control. AIAA/IEEE 4th Digital Avionics Conf., St. Louis, Missouri, Paper 81-2295, November 1981.

3. Sebring, D.L.; and Young, J.T.: Redundancy Management of Skewed and Dispersed Inertial Sensors. AIAA/IEEE 4th Digital Avionics Conf., St. Louis, Missouri, Paper 81-2296, November 1981.

4. Toolan, W.K.; and Zislin, A.M.: Development and Laboratory Test of an Integrated Sensory System (ISS) for Advanced Aircraft. AIAA/IEEE 4th Digital Avionics Conf., St. Louis, Missouri, Paper 81-2297, November 1981.

5. Hopkins, A.L.; Smith, T.B.; and Lala, J.H.: FTMP--A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. Proc. IEEE, vol. 66, no. 10, October 1978, pp. 1221-1239.

6. Larimer, S.J.; and Maher, S.L.: A Continuously Reconfiguring Multi-Microprocessor Flight Control System. AFWAL-TR-81-3070, May 1981.

7. White, J.A. et al.: A Multi-Microprocessor Flight Control System. AFWAL-TR-81-3044, May 1981.

8. "MIL-STD-1750A 16-bit Bipolar Microprocessor Chip Set," Fairchild Advance Information, November 1981.

9. Deutsch, M.S.: Verification and Validation. Chapter 5, pp. 323-408, Software Engineering, R.W. Jensen and C.C. Tonies, eds. Prentice-Hall, Inc., 1979.

10. Rang, E.R.; Gutmann, M.J.; Mulcare, D.B.; and Ness, W.G.: Digital Flight Control Validation Study. Air Force Flight Dynamics Report AFFDL-TR-3076, June 1979.

11. Rang, E.R.: The Use of Finite-State Machines for Describing and Validating Flight Control Systems. Proc., NAECON 1980, Dayton, Ohio, vol. 1, May 1980, pp. 347-353.

12. Lahn, T.G.; and Rang, E.R.: Controlling the Software/Hardware Interface for the Validation of Avionics Systems. AIAA Computers in Aerospace III Conf., San Diego, California. Paper no. 81-2159, October 26-28, 1981, pp. 283-287.

REFERENCES (concluded)

13. Roubine, O.; and Robinson, L.: Special Reference Manual. Third Edition. Stanford Research Institute, Menlo Park, California, Report No. CSG-45, 1977.

14. Military Standard: Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs. MIL-STD-483 (USAF). Notice 2, March 21, 1979.

15. Heninger, K.L.; Kallander, J.W.; Shore, J.E.; and Parnas, D.L.: Software Requirements for the A-7E Aircraft. Naval Research Laboratory, Washington, DC. NRL Report 3876, November 27, 1978.

16. Rang, E.R.; and Gutmann, M.J.: Design and Validating Techniques for Flight Control Systems. Honeywell Systems and Research Center. Report 79SRC92, December 1979.

17. Heninger, K.L.: Specifying Software Requirements for Complex Systems: New Techniques and Their Applications. Proc., Specification of Reliable Software. Cambridge, Massachusetts, April 3-5, 1979, pp. 1-14.

18. Peterson, J.L.: Petri Nets. ACM Computing Surveys, vol. 9, no. 3, September 1977, pp. 223-252.

19. Jack, L.A.; Heimerdinger W. L.; and Johnson, M.D.: Theory of Fault Tolerance. Honeywell Systems and Research Center, Minneapolis, Minnesota, 1974-5 Annual Report, September 1975.

20. Han, Y.W.; and Heimerdinger, W. L.: Theory of Fault Tolerance. Honeywell Systems and Research Center, 1977 Final Report, 77SRC82, Minneapolis, Minnesota, December 1977.

21. Heimerdinger, W. L.; and Fant, K.M.: A Fault Tolerant Assessment of DAIS. Air Force Avionics Laboratory, Wright-Patterson Air Force Base, Ohio, AFAL-TR-79-1007, March 1979.

22. Chow, T.S.: Software Design Modeled by Finite-State Machines. IEEE Trans. Software Eng., vol. SE-4, no. 3, May 1978.

23. Salter, K.G.: A Methodology for Decomposing System Requirements into Data Processing Requirements. Proc. 2nd Conf. on Software Eng., San Francisco, California, October 1976.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-163117		

4. Title and Subtitle	5. Report Date
ADVANCED FLIGHT CONTROL SYSTEM STUDY	November 1982
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
G. L. Hartmann, J. E. Wall, Jr., E. R. Rang, H. P. Lee, R. W. Schulte, and W. K. Ng	82SRC5
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Honeywell Systems and Research Center, 2600 Ridgway Parkway, P.O. Box 312, Minneapolis, Minnesota 55440 — Lockheed-California Co., Burbank, California 91520	NAS4-2876
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report-Final
National Aeronautics and Space Administration, Washington, D.C. 20546	14. Sponsoring Agency Code
	RTOP 512-54-14, 505-34-34

15. Supplementary Notes

NASA Technical Monitor: Larry W. Abbott, Ames Research Center, Dryden Flight Research Facility

16. Abstract

This study defines a flight control system architecture to achieve high integrity fly-by-wire in a cost-effective manner.

Defining an advanced fly-by-wire architecture is a technology integration task. State-of-the-art assessments and trends in the underlying computer, sensing, and actuation areas were used to select from a number of design alternatives.

The recommended architecture emphasizes self-checking microelectronics as a way to reduce the software typically required in redundancy management. This architecture also includes spare sensor and processor elements to permit safe dispatch with failed elements. The second part of this study formulates a methodology capable of demonstrating that the architecture does achieve the required level of performance. This hierarchical methodology ranges from analytical calculations of theoretical system reliablility and formal methods for verifying software to laboratory and iron bird tests and actual flight experiments. The Lockheed S-3A aircraft is discussed as a potential testbed vehicle. The electrical and hydraulic interfaces, together with recommended modifications, are described.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Fly-by-wire Reliability Digital computer architecture S-3A aircraft	Unclassified-Unlimited STAR category 08

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	170	A08

*For sale by the National Technical Information Service, Springfield, Virginia 22161.

End of Document